DTIC FILE COPY

(1)

SECURITY CL

AD-A222 706

ATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFIT/CI/CIA- 90-013D
6a. NAME OF PERFORMING ORGANIZATION AFIT STUDENT AT North Carolina State Univ / 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION AFIT/CIA
6c. ADDRESS (City, State, and ZIP Code)	7b. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB OH 45433-6583
8a. NAME OF FUNDING/SPONSORING ORGANIZATION / 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification) (UNCLASSIFIED)

Conjugate Gradient Methods for Constrained Least Squares Problems

12. PERSONAL AUTHOR(S)
Douglas James

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
THESIS/DISSERTATION	FROM ____ TO ____	1990	135

16. SUPPLEMENTARY NOTATION
APPROVED FOR PUBLIC RELEASE IAW AFR 190-1
ERNEST A. HAYGOOD, 1st Lt, USAF
Executive Officer, Civilian Institution Programs

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

DTIC
S ELECTE D
JUN 15 1990
B

90 06 15 074

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL ERNEST A. HAYGOOD, 1st Lt, USAF	22b. TELEPHONE (Include Area Code) (513) 255-2259 / 22c. OFFICE SYMBOL AFIT/CI

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

Conjugate Gradient Methods for Constrained Least Squares Problems

by

Douglas James

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Department of Mathematics

Raleigh
1990

Approved by:

C. T. Kelley	[illegible]
Carl T. Kelley	Robert E. White
Stephen Wright	[illegible]
Stephen J. Wright	Robert J. Plemmons Chairman, Advisory Committee

Abstract

JAMES, DOUGLAS. Conjugate Gradient Methods for Constrained Least Squares Problems (directed by Robert J. Plemmons). (Barlow, Nichols, and Plemmons)

In 1988, Barlow, Nichols, and Plemmons proposed an order-reducing conjugate gradient algorithm for solving constrained least squares problems. They proved that this method, which we call Algorithm BNP, is superior to p-cyclic SOR in exact arithmetic. (successive overrelaxation)

Here we continue the study of Algorithm BNP. We identify and correct a source of instability in the original algorithm, and develop a parallel version suitable for substructured problems. We prove that BNP is superior to block accelerated overrelaxation (AOR), and establish a connection between BNP and a preconditioned form of the weighting method. We also show that BNP can be viewed as a nullspace method, in which a distinguished choice of the basis for the nullspace is used but never formed. Finally, we exploit this nullspace characterization to extend BNP, producing a class of algorithms we call implicit nullspace methods. These methods allow great flexibility in the choice of preconditioner, and can be used to solve problems for which BNP is not well suited. Like BNP, the extensions are suitable for parallel implementation on substructured problems. Experiments on structural engineering and Stokes Flow models suggest that BNP and its extensions offer a competitive alternative to existing iterative algorithms for solving constrained least squares problems.

The appendix describes a mechanism which can cause the breakdown of incomplete QR factorizations.

Keywords; BNP, SOR, AOR, Theoretical Mathematics, Algorithm, GLSE, Numerical Analysis, Differential Equations, Constrained Minimization Problem,

"I don't know enough," replied the Scarecrow cheerfully. "My head is stuffed with straw, you know, and that is why I am going to Oz to ask him for some brains."

"Oh, I see," said the Tin Woodman. "But, after all, brains are not the best things in the world."

"Have you any?" inquired the Scarecrow.

"No, my head is quite empty," answered the Woodman; "but once I had brains, and a heart also; so, having tried them both, I should much rather have a heart."

L. Frank Baum
The Wizard of Oz

To my brother David.

I remain convinced that
we are the ones who are handicapped.

Acknowledgements

My graduate studies were sponsored by the Department of Mathematical Sciences, US Air Force Academy, CO, and funded by the Air Force Institute of Technology, Wright-Patterson Air Force Base, OH. My research was directed by Robert J. Plemmons under Air Force Office of Scientific Research grant no. AFOSR-88-0285. The Office of Scientific Research, under DURIP grant no. AFOSR-89-0124, also funded the Alliant FX/40 multiprocessor used for all numerical experiments. I have waited a long time for the opportunity to pursue this degree, and am grateful to these agencies for giving me the chance to do so.

My experience at North Carolina State has been a rich and rewarding one; I am indebted to the faculty and staff, especially the members of my committee, for making it so. I especially appreciate all that Professor and Mrs. Plemmons have done to help my family feel welcome. We feel at home here, and we leave Raleigh very reluctantly.

Three friends and mathematicians deserve special thanks. To Professor Jesse Barlow, The Pennsylvania State University: he was probably the first to sense the possibility of a generalization to the original order-reducing conjugate gradient algorithm, and his comments and criticisms on all aspects of this research helped us ask the proper questions. Had our schedules allowed us to work together closely on this problem, this thesis would have been richer. To Professor Gil Strang, Massachusetts Institute of Technology: his book on applied mathematics got me excited about equilibrium equations, while his advice and encouragement helped to keep me going. And to Jim Northrup, formerly my office mate at NC State, and now a professor at Worcester Polytechnic Institute: in the fall of 1987, following my ten year absence from school, he made it possible for me to "hit the ground running" by patiently reacquainting me with the world of computing .

I cannot conclude my acknowledgements without mentioning my family. I thank my parents for gifts and sacrifices beyond words. And to my wife, Pam, and daughter, Bethany, I can only say this: your patience, support, and understanding these last three years has exceeded anything I had a right to expect. Ladies, I have some making up to do.

Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

1. Symbols. When appropriate, the equation number of the first reference to a symbol appears in parentheses. We do not include the following types of symbols in the table: (1) symbols representing iterative approximations (for example, we presume it is clear from the context that $y^{(k)}$ is an approximation of y at the kth iterate); (2) "tilde versions" of primary symbols (in the body of the text, a tilde (˜) over a symbol indicates that the symbol is associated with an unconstrained least squares problem); and (3) symbols used a single time during minor manipulations.

A_1	Preconditioner formed from rows of E and G (3.2)
A_2	Alternate notation for G_2 (3.3)
B_1	INM Preconditioner formed from E and M_1 using interlacing scheme (5.18)
$\hat{B}_1$	INM Preconditioner formed from E and M_1 without using interlacing (5.17)
b	Righthand side of constraint in problem LSE (1.1)
b_0	b augmented with extra component (2.17)
b_1	b augmented with c_1 (3.4)
$\hat{b}$	rb augmented with c in weighting method (3.38)
C	$W_2W_1^{-1}$ in Pwgt (3.47)
c	Constant vector in defect to be minimised (1.1)
c_1	Upper block of c (3.4)
c_2	Lower block of c (3.4)
D	Block diagonal matrix for SOR and AOR splitting (3.35)
D_2	Diagonal block for 2-SOR and 2-AOR (3.36)
D_3	Diagonal block for 3-SOR and 3-AOR (3.36)
D_i^H	ith diagonal block of F_H (2.26)
D_i^V	ith diagonal block of F_V (2.26)
E	Equilibrium matrix (1.1)
E_L	Left block of E_t (3.6)
E_R	Right block of E_t (3.6)
E_s	Stairstep form of E (3.10)
E_t	Trapezoidal form of E (3.6)
E_0	Equilibrium matrix augmented with dependent row (2.17)
$\mathcal{E}$	Young's Modulus (2.12)
F	The matrix G^TG (2.7)
F_H	In Stokes problem, block of F associated with horizontal velocities (2.25)
F_V	In Stokes problem, block of F associated with vertical velocities (2.25)
f	Arbitrary righthand side in linear system (3.35)

$\underline{f}$	Righthand side in conservation of momentum equation (2.13)
f_1	First component of $\underline{f}$ (2.18)
f_2	Second component of $\underline{f}$ (2.19)
G	Coefficient matrix in problem LSE (2.21)
G_H	In Stokes problem, block of G associated with horizontal velocities (2.30)
G_{ij}	Blocks in upper-triangular G; non-standard numbering (3.7)
G_1	Upper block of G (3.1)
G_2	Lower block of G (3.1)
G_V	In Stokes problem, block of G associated with vertical velocities (2.30)
g	Righthand side in conservation of mass equation (2.14)
g_C	function g evaluated at center of stencil (2.15)
h	Righthand side of BNP or INM system (3.21)
h_s	Stokes mesh length (2.15)
I	Identity matrix
K	Block diagonal matrix with 0 and I matrices as the two diagonal blocks (3.4)
$\hat{K}$	Rightmost block of K (3.16)
L	Block lower triangular matrix for SOR and AOR splitting (3.35)
M_1	Augmentation matrix used to form B_1 (5.17)
m_1	Number of rows in E (1.1)
m_2	Number of rows in G (1.1)
N	Basis matrix for the nullspace of E (4.15), (5.2)
N_f	Fundamental basis matrix for nullspace of E (5.4)
n	Number of unknowns in problem LSE (1.1)
n_s	In Stokes problem, number of cells on each side of unit square
P	Permutation matrix (3.9)
P_L	Left block of P (3.9)
P_R	Right block of P (3.9)
p	pressure (2.13)
$\mathcal{P}(y)$	quadratic form in constrained minimization formulation (2.4)
q_k	Direction vector for correcting y in BNP (3.28)
r	Residual $r = c - Gy$ (2.6)
r_1	Upper block of r; unknown in BNP (3.4)
r_2	Lower block of r (3.4)
S_k	kth Krylov subspace (4.14)
s	The vector $-G^T c$ (1.1)
s_k	BNP direction vector (3.27)
t	Thickness of element (2.12)
U	Block upper triangular matrix for both the SOR and AOR splittings (3.35)
$\underline{v}$	Flow velocity vector (2.13)
v_i	ith component of $\underline{v}$ (2.18)
W	τE augmented with G in weighting method (3.38)
W_1	upper block of W (3.43)
W_2	lower block of W (3.44)
w	unknown in various preconditioned systems (3.47)
w_L	Upper block of w (4.33)
w_R	Lower block of w (4.33)
x	As a vector, the unknown in INM (4.2); as a scalar, horizontal coordinate direction (2.18)
Y	The matrix $A_2 A_1^{-1} \hat{K}$ (3.20)
y	Unknown in problem LSE (1.1)
y_B	Velocity on a fictitious boundary edge (2.22)
y_C	Component of y in center of stencil (2.15)
y_E	Component of y along east edge of stencil (2.15)
y_L	Upper block of y (5.12)
y_N	Component of y along north edge of stencil (2.15)
y_p	Particular solution to constraint $Ey = b$ (4.17)

y_R	Lower block of y (5.12)
y_S	Component of y along south edge of stencil (2.15)
y_W	Component of y along west edge of stencil (2.15)
z	As a vector, unknown in arbitrary linear system (3.35); as a scalar, vertical coordinate direction (2.10)
z_3	λ augmented with r_1 (3.4)
β	AOR acceleration parameter (3.37)
γ_k	BNP scale factor (3.27)
λ	Lagrange multiplier in Kuhn-Tucker formulation (2.6)
μ	Lagrange multiplier in saddle point formulation (2.8)
μ_E	Component of μ along east edge of stencil (2.20)
μ_W	Component of μ along west edge of stencil (2.20)
ν	Poisson's Ratio (2.12)
ν_k	BNP defect (3.29)
τ	weighting parameter in weighting method (3.38)
ω	Relaxation parameter in SOR and AOR (3.35)
∇	Gradient operator (2.13)
$\nabla\cdot$	Divergence operator (2.14)
∇^2	Laplacian operator (2.13)

2. Abbreviations. The description of each abbreviation includes a reference to the section in which the item is first discussed in detail.

AOR	Accelerated Over-relaxation (§3.3)
BNP	Order-reducing conjugate gradient algorithm due to Barlow, Nichols, and Plemmons (§3.2)
INM	Implicit Nullspace Method (§5.2)
INMF	Implicit Nullspace Method with preconditioner based on F (§5.3)
INMG	Implicit Nullspace Method using rows of G (§5.3)
INMGI	Implicit Nullspace Method using rows of G and I (§5.3)
INMI	Implicit Nullspace Method using rows of I (§5.3)
IQR	Incomplete QR (§9.1)
LSE	Equality constrained least squares problem (§2.1)
PWgt	Preconditioned Weighting Algorithm (§3.3)
SOR	Successive Over-relaxation (§3.3)
WLS	Weighted Least Squares (§3.3)
2-AOR	Two-block Accelerated Over-relaxation (§3.3)
2-SOR	2-cyclic Successive Over-relaxation (§3.3)
3-AOR	Three-block Accelerated Over-relaxation (§3.3)
3-SOR	3-cyclic Successive Over-relaxation (§3.3)

1. Introduction

In this thesis, we consider the solution of the equality constrained least squares problem, or problem LSE: given an $m_1 \times n$ matrix E, an $m_2 \times n$ matrix G, an $m_1 \times 1$ vector b, and an $m_2 \times 1$ vector c,

$$\text{minimize } \|Gy - c\|_2 \quad \text{such that} \quad Ey = b. \tag{1.1}$$

We place conditions on the problem guaranteeing a unique solution, and presume that the matrices are large and sparse, so that it is plausible to consider iterative algorithms. All assumptions are realistic for a wide range of important applications.

We will also need a number of equivalent formulations of the problem. The so-called **Kuhn-Tucker formulation** is the starting point for each of the algorithms we discuss, while the **constrained minimization** and **saddle point** forms is the natural setting for many physical applications.

The motivating example for our work is the static analysis of engineering structures: given a large structure subjected to an external load, find the internal forces at equilibrium. When the problem is modelled using the **force method** (see, for example, [17]), the constraint $Ey = b$ captures the fact that the forces sum to zero at any node in the structure (the **equilibrium condition**). We then look for the unique set of internal forces which minimizes a type of potential energy subject to this constraint. This application is but one example of a more general physical principle at work: minimizing an energy functional subject to an equilibrium constraint is a central idea throughout the physical sciences (see Strang [34], [35]), so problem LSE in its various equivalent forms has a wide range of applications. We use as a second example a discretization of a Stokes Flow problem leading to a saddle point system.

Our study of problem LSE and its equivalent forms centers around an algorithm first proposed and analyzed by Barlow, Nichols, and Plemmons [4]. The derivation of this algorithm, which we call **Algorithm BNP**, starts with a repartitioned form of the Kuhn-Tucker equations which has square, nonsingular, and easily invertible diagonal blocks. The authors rewrite this system by applying block Gauss elimination, producing a symmetric positive definite sub-problem with the $n-m_1$ leading components of the residual $r = c - Gy$ as the unknowns. They then apply a variation of the conjugate gradient algorithm to this sub-problem, generating at each iteration an approximation to the original unknown y. The method is **order- or dimension-reducing** in the sense that the sub-problem has fewer unknowns than the original problem in y.

Barlow, Nichols, and Plemmons prove in [4] that Algorithm BNP is superior to so-called **p-cyclic SOR methods** in a certain well-defined sense. We extend this result, proving the algorithm is also superior to a two-parameter generalization of SOR known as **Block Accelerated Overrelaxation** or AOR. We also establish a connection between BNP and a preconditioned version of the classical **weighting method**. In addition, we identify and correct a source of inaccuracy in the original version of algorithm BNP.

Our main purpose, however, is to extend the algorithm to two types of problems for which Algorithm BNP is not well suited. In theory, BNP requires fairly mild assumptions on problem LSE. In practice, however, implementation is difficult unless the matrix G has full column rank. This assumption holds for many but not all applications (it fails to hold, for example, if any of the elements in the structures problem fails to behave elastically). The algorithm may also be difficult to apply to problems expressed in saddle point form (e.g. Stokes flow). We propose an extension to BNP which can overcome these limitations.

The key to the extension is recognizing a connection between algorithm BNP and the classical nullspace method. The latter begins with some convenient particular solution y_p to the constraint, and a matrix N whose columns form a basis for the nullspace of E. One then seeks a coordinate vector x such that $y = y_p + Nx$ is the solution to the constrained problem. We show that BNP may be viewed as a variation of the nullspace method, in which distinguished choices of y_p and N are used but never formed (we will call such a method an **implicit nullspace method**, or INM). The basis matrix N is seen to be acting as a preconditioner for a set of normal equations in factored form. We generalize algorithm BNP by producing implicit nullspace methods for other choices of N. The extension preserves the spirit of BNP (in particular, the algorithms are order-reducing), but is somewhat more flexible: it becomes relatively easy to construct preconditioners for problems in which G lacks full column rank, as well as problems in saddle point form. Both BNP and the more general implicit nullspace algorithms can be implemented in parallel when the matrices reflect a substructuring (domain decomposition) of the physical model.

Chapter 2 provides an overview of the properties of problem LSE and its equivalent forms. We also show how both the structures and Stokes Flow problems can be expressed in terms of these forms, and discuss the special structure of the underlying matrices for substructured problems. Chapter 3 provides an overview of each of the four basic iterative algorithms we discuss: p-cyclic SOR, block AOR, a preconditioned form of the weighting method, and of course Algorithm BNP itself. We include a detailed look at the construction of the modified Kuhn-Tucker equations needed to implement all of these algorithms. In chapter 4 we compare and contrast the methods: we summarize the results in [4] concerning BNP and p-cyclic SOR; we extend these results to show that BNP is superior to block AOR; and we show that BNP can be viewed as the limiting case of the preconditioned weighting method.

The extension of BNP is the topic of chapter 5: we establish the relationship between BNP and the nullspace method, then exploit this connection to derive the more general implicit nullspace methods. We then provide a menu of possible INM algorithms. In chapter 6 we summarize the results of our numerical experiments. We include tests on a variety of structural engineering problems, including problems which violate the assumptions of elasticity. We also describe the performance of various forms of INM on a Stokes Flow problem. In each case, we include results of experiments involving parallel versions of the algorithms.

We offer some concluding remarks in chapter 7. Finally, in the appendix, we describe a mechanism which can cause the breakdown of certain types of incomplete QR factorizations proposed in the literature.

2. Formulation of the Problem

This chapter is an overview of the equality constrained least squares problem. In §1.1 we mention some of the basic properties of problem LSE, state the assumptions under which we proceed, and express the problem in four equivalent forms. The material in §1.1 is largely a distillation of more complete discussions in Bjorck [7], Golub and van Loan [12], and Strang [35]. The next two sections include descriptions of two physical applications: the static analysis of structures (§1.2) and a finite difference discretization of a Stokes Flow problem (§1.3). In §1.4, we use these two examples to explain the effect of substructuring (domain decomposition) on the structure of the underlying matrices.

2.1 Problem LSE and Equivalent Forms

Recall from the introduction the statement of problem LSE: given an $m_1 \times n$ matrix E, an $m_2 \times n$ matrix G, an $m_1 \times 1$ vector b, and an $m_2 \times 1$ vector c,

$$\text{minimize } \|Gy - c\|_2 \quad \text{such that} \quad Ey = b. \tag{2.1}$$

We make two plausible assumptions before proceeding:

H1: E has full row rank, so b is in the range of E and the problem has at least one solution; and

H2: the nullspaces of E and G intersect trivially (or, equivalently, $\begin{bmatrix} E \\ G \end{bmatrix}$ has full column rank), guaranteeing that the solution is unique.

Two types of manipulations will prove especially useful. Given any non-singular matrix B, we can replace the constraint $Ey = b$ in problem LSE with the new constraint $BEy = Bb$. This means we can apply Gauss elimination or orthogonal reduction to E and b without affecting the solution y; we can use this fact to reduce E to some convenient "canonical" form. Similarly, let Q be an orthogonal matrix. Then, since premultiplication by Q preserves the 2-norm, we can replace the quantity $(Gy - c)$ in LSE with $(QGy - Qc)$. Thus we can apply orthogonal rotations (but not Gauss elimination) to the rows of G and c without affecting the solution. Additionally, we can scale $(Gy - c)$ by any constant multiple of the identity matrix.

We will also make frequent use of the three equivalent formulations of LSE described below.

Constrained Minimization Problem. Note that

$$\|Gy - c\|_2^2 = y^T G^T G y - 2y^T G^T c + c^T c, \tag{2.2}$$

and that the term $c^T c$ has no effect on the value of y at which the quantity is minimized. With this in mind, define

$$\mathcal{P}(y) = y^T G^T G y - 2y^T G^T c, \tag{2.3}$$

noting that y satisfies problem LSE if and only if y is the solution of the constrained minimization problem:

$$\text{minimize } \mathcal{P}(y) \quad \text{such that} \quad Ey = b. \tag{2.4}$$

Kuhn-Tucker Formulation. Let $r = c - Gy$ represent the residual vector we seek to minimize. Introduce the $m_1 \times 1$ Lagrange multiplier λ (one component for each constraint), and define the functional

$$\phi(y, \lambda) = \mathcal{P}(y) + \lambda^T(Ey - p). \tag{2.5}$$

Here $\mathcal{P}$ is as in equation (2.3). Under assumptions H1 and H2, the solution y to problem LSE is part of the ordered pair (y, λ) defining the unique saddle point of this functional. One can find this saddle point by solving the so-called Kuhn-Tucker equations:

$$\begin{bmatrix} E & 0 & 0 \\ G & I & 0 \\ 0 & G^T & E^T \end{bmatrix} \begin{bmatrix} y \\ r \\ \lambda \end{bmatrix} = \begin{bmatrix} b \\ c \\ 0 \end{bmatrix}. \tag{2.6}$$

The first block equation in this system, which is just the constraint $Ey = b$, results from differentiating ϕ with respect to each component of λ, and setting the result equal to zero. The second equation simply defines the residual r. The third equation comes from differentiating ϕ with respect to each component of y. The Kuhn-Tucker formulation is the starting point for most of the algorithms of concern to us.

Saddle Point Formulation. Let $F = G^TG$ and $s = -G^Tc$. Note that the functional $\mathcal{P}$ defined in equation (2.3) can then be rewritten as

$$\mathcal{P}(y) = y^TFy + 2y^Ts. \tag{2.7}$$

Introduce the Lagrange multiplier μ, and define the functional

$$\psi(y, \lambda) = \mathcal{P}(y) - \mu^T(Ey - p). \tag{2.8}$$

Once again the solution to LSE is associated with the saddle point of this functional; thus one can find the solution to problem LSE by solving the linear system which defines this saddle point:

$$\begin{bmatrix} -F & E^T \\ E & 0 \end{bmatrix} \begin{bmatrix} y \\ \mu \end{bmatrix} = \begin{bmatrix} s \\ b \end{bmatrix}. \tag{2.9}$$

We chose the signs in the definitions of μ, s and ψ so the matrix E is the same in all four formulations. The multiplier μ is related to λ in the Kuhn-Tucker equations by $\mu = -\lambda$.

2.2 First Example: Static Analysis of Engineering Structures

The motivating example for our work is the static analysis of engineering structures: given a physical structure subject to an external load, find the internal forces at equilibrium. In this section, we use a simple example to show how to model this problem as a constrained least squares problem. We employ the so-called **force method** (see, for example, Heath, Plemmons, and Ward [17], Przemieniecki [32], or virtually any text on structural analysis). Much of the material below is a summary of the discussion in section 2.4 of Strang [35].

Consider the truss depicted in figure 2.1. The elements in the structure are rigid bars. We assume the bars cannot bend; longitudinally, they behave as if they were very stiff springs, reacting to external loads by compressing or elongating. Thus, associated with each element i there is a single unknown y_i representing the internal force in that element when the structure as a whole is at equilibrium. We establish the convention that a positive y_i represents an element in tension.

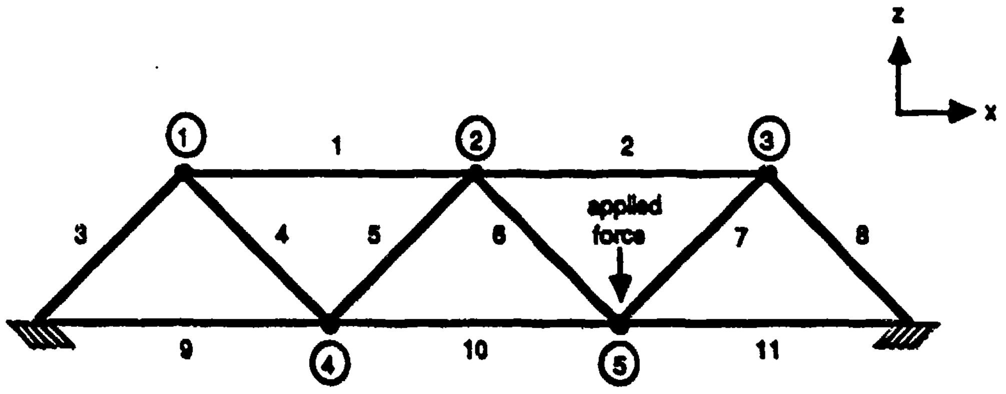


Figure 2.1: Pin-jointed Rigid Bar Truss

The elements are connected to each other by pins; the locations of these pins are called **nodes**. We assume the problem is modelled so that external forces act only at the nodes; in this example, there is a single unit force acting downward on node 5. The nodes at the southwest and southeast corners of the truss are supported or "grounded" in the sense that pin joints bolt them to an immobile base (this means we are not free to specify an external loading at these nodes; the force on a fixed node is that necessary to prevent the node from moving). The other five nodes are free to move in the x–z plane as the elements lengthen or contract under the load. We assume, however, that the **displacements** at these free nodes are very small. We also presume that the structure is **stable**: there are enough properly placed elements to prevent the structure from collapsing on itself, and it is supported in such a way that there is no set of external loads which causes a rigid motion.

If the structure is at equilibrium, then the forces acting at each node must sum to zero (the **equilibrium condition**). In the example, we therefore begin by writing a force balance in both the x and z coordinate directions for each of

$$\begin{bmatrix} -1 & & \sqrt{2} & -\sqrt{2} & & & & & & & \\ & & \sqrt{2} & \sqrt{2} & & & & & & & \\ 1 & -1 & & & \sqrt{2} & -\sqrt{2} & & & & & \\ & & & & \sqrt{2} & \sqrt{2} & & & & & \\ & 1 & & & & & \sqrt{2} & -\sqrt{2} & & & \\ & & & & & & \sqrt{2} & \sqrt{2} & & & \\ & & & \sqrt{2} & -\sqrt{2} & & & & 1 & -1 & \\ & & & -\sqrt{2} & -\sqrt{2} & & & & & & \\ & & & & & \sqrt{2} & -\sqrt{2} & & & 1 & -1 \\ & & & & & -\sqrt{2} & -\sqrt{2} & & & & \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Figure 2.2: Equilibrium Constraint $Ey = b$ for Pin-jointed Truss

the five free nodes. At node 5, for instance, the equilibrium condition gives us the two equations

$$\begin{aligned} +\sqrt{2}y_6 - \sqrt{2}y_7 + y_{10} - y_{11} &= 0 \qquad (x \text{ direction}) \\ -\sqrt{2}y_6 - \sqrt{2}y_7 &= 1 \qquad (y \text{ direction}). \end{aligned} \tag{2.10}$$

We now write the collection of all ten equations in matrix form: each of the five free nodes contributes a block of two rows (one for each direction in which the node is free to move). The result is the constraint $Ey = b$ in problem LSE. The full system for the truss in figure 2.1 appears in figure 2.2.

The vector b stores the specified external force vector. The matrix E, commonly called the equilibrium matrix, holds the non-zero coefficients in the force balance equations. It captures the shape and connectivity of the structure; its entries are independent of material properties. The matrix E has one other important property: if the structure is stable, then the equilibrium matrix has full row rank (see Strang [35]). Thus the equilibrium constraint for a stable structure satisfies hypothesis H1. Because each node is attached to only a small number of elements, E is extremely sparse when the structure is large.

More generally, a finite element model of a structure may be composed of elements which allow more freedom and complexity than rigid elastic bars (see, for example, Przemieniecki [32]). Figure 2.3 depicts some typical examples: a pair of two-dimensional planar elements (in this case, a right triangle and a square), and a solid tetrahedral element. In these examples we assume the only independent internal forces are those shown in the sketches, and we take the corners of the elements to be the nodes. Other more complex models are possible (e.g. elements which account for the effects of bending, supports which restrict nodes in some but not all directions, etc.), but we limit our examples to the types of elements which appear in our test problems.

In any case, regardless of the types of elements in the structure, one still constructs the equilibrium constraint $Ey = b$ by writing force balance equations at each free node. Now, however, each element generates a *block* of columns in the equilibrium matrix E (one column for each independent internal force). Thus, for example, the equilibrium matrix for a two-dimensional structure modelled with planar triangular elements will have three columns per element, and two rows per free node. The matrix E will be sparse, and will have full row rank when the structure is stable. Unlike the matrix in figure 2.2, however, the equilibrium matrix will often have far more columns than rows (see the test problems in chapter 6, where m_1 ranges from roughly $\frac{1}{3}$ to $\frac{1}{2}$ the size of n_e).

Of course, the equilibrium condition does not tell the whole story. Mathematically, there will generally be an infinite number of vectors y satisfying the constraint $Ey = b$. Physically, we have not yet accounted for the material

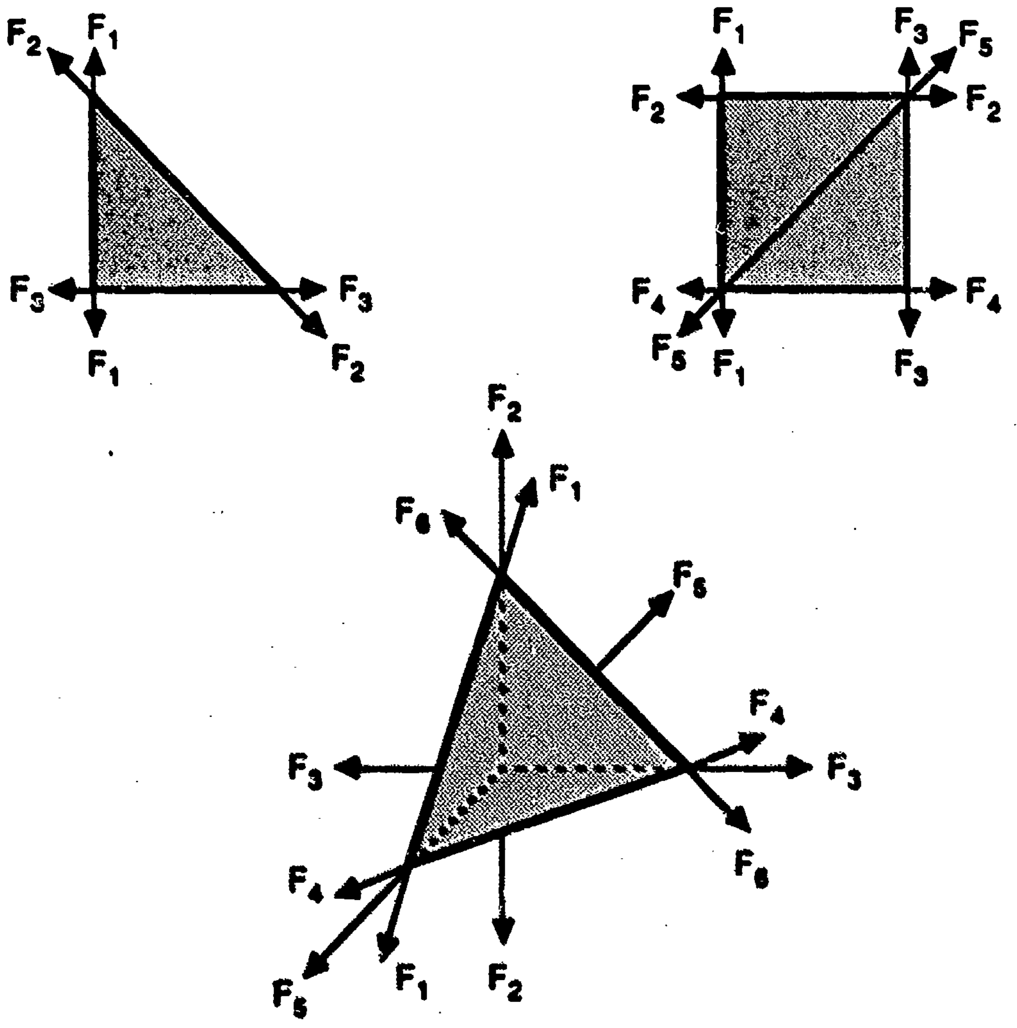


Figure 2.3: Typical Planar and Solid Elements

properties of the elements in the structure. A symmetric non-negative definite matrix F, called the element flexibility matrix, captures the material properties. The solution we seek is the set of internal forces y which minimize complementary energy subject to the equilibrium constraint (see, for example, Strang [35]):

$$\text{minimize } y^T F y \quad \text{such that} \quad Ey = b. \tag{2.11}$$

Thus we can express the static structures problem as a constrained minimization problem. Unless the elements are pre-stressed, the vectors c and $s = -G^T c$ given in equations (2.1) and (2.7) are zero in this application. The components of the vector $-\lambda$ in the Kuhn-Tucker formulation represent the small displacements of the nodes from their neutral (unloaded) positions.

For a truss composed of rigid elastic bars, F is a diagonal matrix with positive diagonal entries determined by Hooke's Law (the entry in position (i, i) of F is the reciprocal of the "spring" constant for element i; again, see Strang [35]). G, of course, is a diagonal matrix with diagonal entries equal to the square roots of the corresponding entries in F. For more general elements, F is a block diagonal matrix, with one small block for each element in the structure. The block associated with the right triangular planar element in figure 2.3, for example, is the 3×3 matrix

$$F_\Delta = \frac{2}{\mathcal{E}t} \begin{bmatrix} 1 & \frac{(1-\nu)}{\sqrt{2}} & -\nu \\ \frac{(1-\nu)}{\sqrt{2}} & 2 & \frac{(1-\nu)}{\sqrt{2}} \\ -\nu & \frac{(1-\nu)}{\sqrt{2}} & 1 \end{bmatrix}, \tag{2.12}$$

where the physical constants $\mathcal{E}$, ν, and t represent Young's Modulus, Poisson's Ratio, and the thickness of the element respectively (see [32]). The matrix is positive definite for $0 < \nu < 1$ (the value $\nu = 0.3$ is typical). The blocks

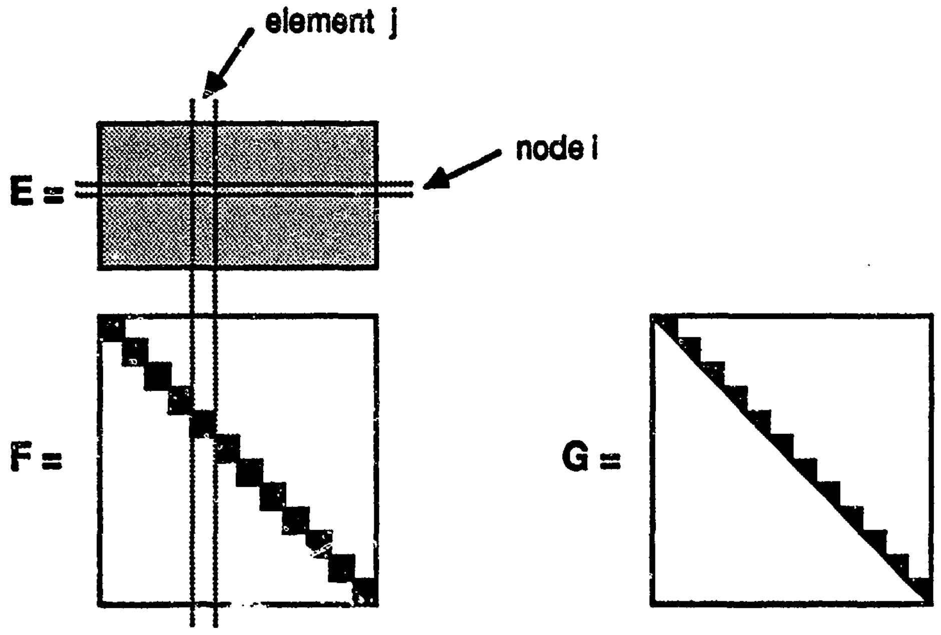


Figure 2.4: Matrices for Static Structures Problem

associated with the planar square element and tetrahedral element (5×5 and 6×6 respectively) are defined by similar formulas. In all cases, the matrix F is symmetric positive definite when all elements behave elastically. Thus, the Cholesky factor of F, which is block diagonal with upper triangular blocks, can be used as the matrix G (figure 2.4). Moreover, since G is square and non-singular for elastic problems, hypothesis H2 from the previous section is satisfied trivially.

At a critical value of ν, however, an element no longer behaves elastically, and the block of F associated with the element becomes singular. This critical value ($\nu = 1$ for the planar elements, $\nu = \frac{1}{2}$ for the tetrahedron) represents the point at which an element no longer changes area (or volume) when subjected to a load (see [32]). At the critical value, the block is still symmetric and non-negative definite. Thus, when such blocks are present in F, we can still find G such that $F = G^T G$, but now the matrix G lacks full column rank. While the linear theory described above no longer provides an adequate model of the physics in this case, we will use these critical values to simulate "damaged" elements in the structure, giving us an opportunity to generate test problems which are difficult to solve using algorithm BNP as originally derived.

We conclude this section by mentioning that the force method described above is one of two approaches to solving the static structures problem. A second formulation, the so-called **displacement method**, solves an unconstrained least squares problem for the displacements, and then recovers the force vector y (see Strang [35] for a careful discussion of the connection between the two methods). For many reasons, the displacement method is by far the more common approach used to solve structures problems. The force method formulation described here, however, has some advantages. In particular, it explicitly separates calculations involving the geometry of the structure (the constraint) from those involving the material properties (the energy functional), and can therefore be useful when analyzing a fixed structure while varying the material properties (see, for example, Batt and Gellin [1]).

2.3 Second Example: Stokes Flow

The static analysis of engineering structures provides but one example of a more general physical principle at work: minimizing an energy functional subject to an equilibrium constraint is a central idea throughout the physical sciences (see Strang [34], [35]). A second example, leading naturally to a saddle point problem, is the standard Stokes model for steady, very viscous flows. If we consider only the constant density case, and scale out the Reynolds number, the equations describing the flow are given by

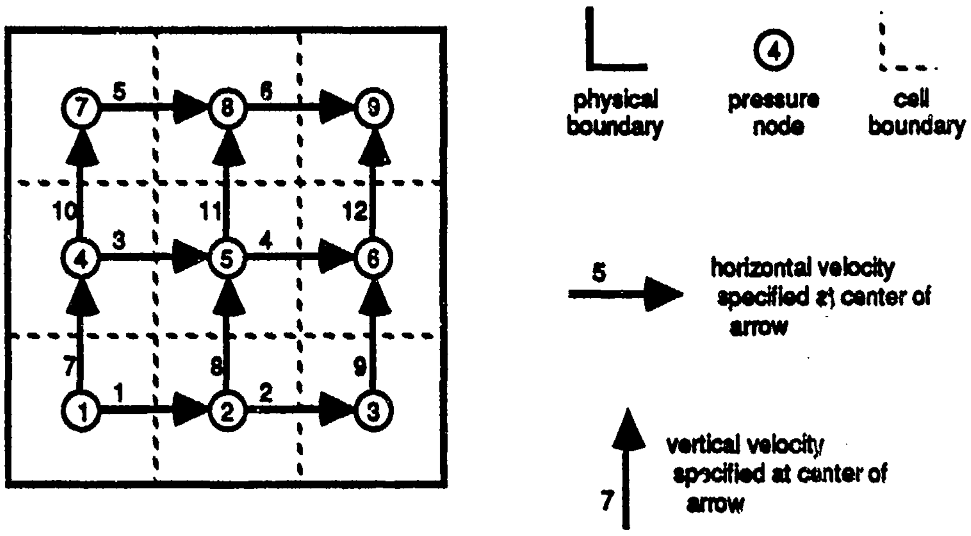
$$\nabla^2 \underline{v} - \nabla p = \underline{f} \tag{2.13}$$

$$\nabla \cdot \underline{v} = g. \tag{2.14}$$

The quantity $\underline{v}$ represents the continuous velocity vector, with one component for each coordinate direction. The scalar p represents pressure. The first equation comes from conservation of momentum; the vector-valued forcing term $\underline{f}$ reflects external forces such as gravity. The Laplacian operator ∇^2 acts on each component of $\underline{v}$ separately. The second equation reflects conservation of mass; in the absence of sources or sinks, the righthand side g is zero.

We must also specify appropriate boundary conditions. We consider only the Dirichlet boundary condition $\underline{v} = \underline{v}_0$ (the case $\underline{v} = 0$, which is physically plausible for viscous flows at a solid boundary, is the so-called "no-slip" boundary condition). Note that only the derivative (gradient) of pressure appears in the differential equations. Thus, in the absence of boundary conditions on pressure, we can determine pressure only up to a constant.

When we discretize the continuous problem, we expect to obtain the saddle point form (2.9). The matrix E will approximate the divergence operator, and the conservation of mass equation will become the constraint $Ey = b$. The vector b will reflect boundary contributions as well as the forcing term g. The components of the Lagrange multiplier μ will represent pressure. Since the negative of the gradient is the adjoint of the divergence, we will find that $E^T\mu$ will approximate $-\nabla p$. The symmetric positive definite matrix F will come from a discretization of the negative of the Laplacian operator ∇^2, and s will



7
5
8
6
9
10
11
12
4
3
5
4
6
7
8
9
1
1
2
2
3
physical boundary
4
pressure node
cell boundary
5
horizontal velocity specified at center of arrow
vertical velocity specified at center of arrow
7

Figure 2.5: Marker-and-Cell Discretization of Unit Square (n_s=3)

reflect both boundary contributions and the forcing function $\underline{f}$.

The test problems in chapter 6 are based on a marker-and-cell or MAC finite difference discretization first studied by Harlow and Welch [16] (see also Hall [15]). Here we consider the MAC method for the Dirichlet problem on the unit square. Let x and z be the horizontal and vertical coordinate directions, and begin by dividing the domain into cells as shown in figure 2.5. For simplicity we consider a regular partitioning of the domain: let n_s be the number of cells in each coordinate direction, and $h_s = 1/n_s$ the horizontal and vertical length of each cell. We specify the pressure μ at nodes placed at the center of each cell. Now connect the pressure nodes with directed line segments. The components of the discrete velocity vector y are specified at the midpoints of these directed line segments: horizontal components of velocity on horizontal line segments (edges), and vertical components of velocity on vertical edges.

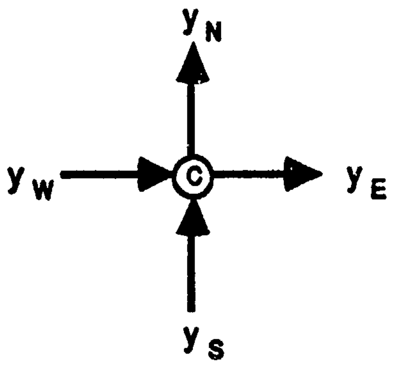


Figure 2.6: MAC Stencil: Divergence Operator

Choose a convenient numbering of the nodes and edges. For now, we number all the horizontal edges before the vertical ones, and we number both nodes and edges from left to right, beginning at the bottom. Note that the boundary of the grid does not reach the physical boundary; instead, it is offset by a distance of $h_s/2$.

We discretize the conservation of mass equation $\nabla \cdot \underline{y} = g$ by writing a flow balance equation at each *node* in the grid, much as we did for the structures problem in the previous section. Letting the subscripts N, S, E, W, and C represent the north, south, east, west, and center positions on a stencil (figure 2.6), centered differences at each interior pressure node produces the equation

$$(y_E - y_W) + (y_N - y_S) = h_s g_C. \tag{2.15}$$

At boundary nodes, one or more values of y are given by the boundary data, and must be brought to the righthand side of the flow balance equation. For example, at nodes on the interior of the the west wall of the grid, y_W is known, and the equation becomes

$$y_E + y_N - y_S = h_s g_C + y_W. \tag{2.16}$$

$$\left[\begin{array}{cc|cc|cc|ccc|ccc}
1 & & & & & & 1 & & & & & \\
-1 & 1 & & & & & & 1 & & & & \\
 & -1 & & & & & & & 1 & & & \\
\hline
 & & 1 & & & & -1 & & & 1 & & \\
 & & -1 & 1 & & & & -1 & & & 1 & \\
 & & & -1 & & & & & -1 & & & 1 \\
\hline
 & & & & 1 & & & & & -1 & & \\
 & & & & -1 & 1 & & & & & -1 & \\
 & & & & & -1 & & & & & & -1
\end{array}\right]$$

Figure 2.7: Matrix E_0 for MAC Grid (n_s=3)

If we now write the collection of all such equations as a linear system, we obtain a matrix equation of the form

$$E_0 y = b_0. \tag{2.17}$$

The matrix E_0 for the case n_s=3 is shown in figure 2.7; the pattern for larger n_s is similar. But E_0 is not quite the equilibrium matrix we seek. We will produce the matrix E and the associated constraint $Ey = b$ after a modification described later in this section.

Before completing the construction of the constraint, we address the discretization of the conservation of momentum equation (2.13). Recognize first that the equation is a vector equation with components in the two coordinate directions:

$$\begin{aligned} \nabla^2 v_1 - p_x &= f_1 \qquad (x \text{ component}) & (2.18) \\ \nabla^2 v_2 - p_z &= f_2 \qquad (z \text{ component}). & (2.19) \end{aligned}$$

Here v_1 and v_2 represent the x and z components of the continuous velocity vector $\underline{v}$; p_x and p_z are the derivatives of pressure with respect to x and z

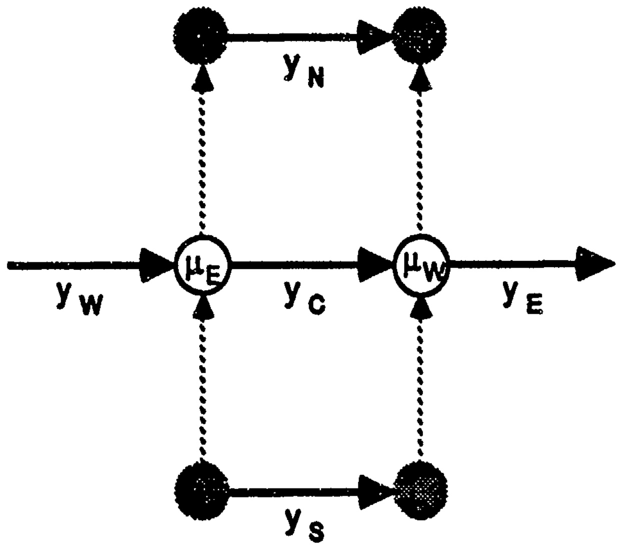


Figure 2.8: MAC Stencil: Horizontal Momentum

respectively; and f_1 and f_2 are the components of the vector-valued forcing term $\underline{f}$.

We discretize the conservation of momentum equation by writing a difference equation at the center of each *edge* of the grid: on horizontal edges, we use the horizontal equation (2.18), while on vertical edges we use (2.19). Consider, for example, a typical horizontal edge with associated discrete velocity component y_C. Recalling that the Lagrange multiplier μ represents pressure at the nodes, and using the subscripts N, S, E, W, and C as we did above, we work with the stencil depicted in figure 2.8. Using centered differences, and multiplying through by h_s (not h_s^2), we obtain the equation

$$-\frac{1}{h_s}(4y_C - y_N - y_S - y_E - y_W) - (\mu_E - \mu_W) = h_s f_1, \tag{2.20}$$

where f_1 is evaluated at the midpoint of the edge associated with y_C. A similar equation results from discretizing equation (2.20) at the center of interior

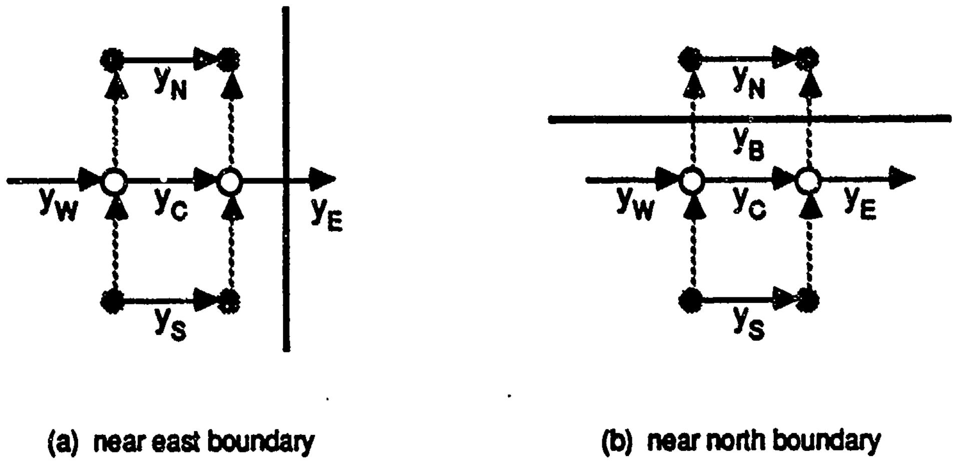


Figure 2.9: MAC Boundary Corrections: Horizontal Momentum

vertical edges.

Because the boundary of the grid and the physical boundary do not coincide, the treatment of the edges near the boundary is a bit delicate. There are two cases to consider. The first case involves an edge y_C adjacent and *perpendicular* to the boundary, such as the edge near the east boundary shown in figure 2.9a. This situation presents no special problems: a fictitious edge (y_E in the example) has a known value given by the boundary data. We simply substitute this known value into equation (2.20) (or the equivalent equation in the vertical direction if appropriate), and move it to the right-hand side. For the example we obtain

$$-\frac{1}{h_s}(4y_C - y_N - y_S - y_W) + (\mu_E - \mu_W) = h_s f_1 - \frac{1}{h_s} y_E. \qquad (2.21)$$

The second case involves an edge y_C adjacent and *parallel* to the boundary, such as the edge near the north boundary in figure 2.9b. Here the fictitious edge (y_N in the example) lies outside the physical boundary, so its value is not specified by the boundary data. Instead, we know the value at a fictitious edge y_B lying on the boundary midway between y_C and y_N. We now interpolate to obtain an approximate value for the fictitious edge: $y_B = \frac{1}{2}(y_N + y_C)$, or

$$y_N = 2y_B - y_C. \tag{2.22}$$

Now substitute y_N into equation (2.20) and rearrange to get what we need:

$$-\frac{1}{h_s}(4y_C - y_S - y_E - y_W) - (\mu_E - \mu_W) = h_s f_1 - \frac{2}{h_s} y_N. \tag{2.23}$$

Of course edges near corners of the physical domain require both types of corrections.

If we write the collection of all the edge equations as a linear system, we obtain a matrix equation of the form

$$-Fy + E_0^T \mu = s, \tag{2.24}$$

where E_0 is precisely the matrix defined in equation (2.17). The matrix F contains two copies of the discretized Laplacian:

$$F = \frac{1}{h_s}\begin{bmatrix} F_H & \\ & F_V \end{bmatrix}, \tag{2.25}$$

where F_H and F_V are associated with the horizontal and vertical edges respectively. More precisely, F_H and F_V are the block tridiagonal matrices

$$F_H = \begin{bmatrix} D_1^H & -I & & \\ -I & D_2^H & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & D_{n_s}^H \end{bmatrix} \qquad F_V = \begin{bmatrix} D_1^V & -I & & \\ -I & D_2^V & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & D_{n_s-1}^V \end{bmatrix}, \tag{2.26}$$

where $D_1^H = D_{n_s}^H$ is the $(n_s - 1)\times(n_s - 1)$ symmetric tridiagonal matrix with 5's on the main diagonal and 1's on the sub- and superdiagonals, $D_2^H = \ldots = D_{n_s-1}^H$ is the $(n_s - 1)\times(n_s - 1)$ symmetric tridiagonal matrix with 4's on the main diagonal and 1's on the sub- and superdiagonals, and D_i^V is the $n_s \times n_s$ matrix

$$D_i^V = \begin{bmatrix} 5 & -1 & & & & & \\ -1 & 4 & -1 & & & & \\ & -1 & 4 & -1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -1 & 4 & -1 & \\ & & & & -1 & 4 & -1 \\ & & & & & -1 & 5 \end{bmatrix} \tag{2.27}$$

for $i = 1, \ldots, n_s - 1$. In all cases, I represents the appropriately sized identity matrix. While it's helpful to appreciate the structure of the matrix F, in general it need not be formed explicitly. We can compute a matrix-vector product of the form Fw in terms of the stencils defining the action of F on an arbitrary vector w.

At this point, our discretized system has the form

$$\begin{bmatrix} -F & E_0^T \\ E_0 & 0 \end{bmatrix} \begin{bmatrix} y \\ \mu \end{bmatrix} = \begin{bmatrix} s \\ b_0 \end{bmatrix}. \tag{2.28}$$

This is certainly a saddle point system, but the matrix E_0 does not satisfy the required hypotheses. Refer to figure 2.7, and note that E_0 lacks full row rank (the sum of the rows of E_0 is the zero vector). This is no surprise: since we are assuming Dirichlet boundary conditions, every edge enters one node and exits another, so every column of E_0 contains a single entry equal to -1, a single entry equal to $+1$, and no other non-zero entries. Up to discretization error, the same is true of the righthand side vector b_0; this is a straightforward consequence of Green's Theorem. We need to produce a constraint $Ey = b$ such that E has full row rank. We can do this in the obvious way, by deleting any convenient row of E_0 and the corresponding component of b_0 (more precisely, by annihilating the row using Gauss elimination or orthogonal reduction). But we must examine the effect of this change on the discrete conservation of momentum equation.

To do this, recall that the continuous problem determines pressure only up to a constant. Thus if we specify the value of pressure at any single node, we should expect to determine the pressure field uniquely. For convenience, set the last component of μ to zero. Let e^T be the corresponding row of E_0, and partition E_0 and μ into

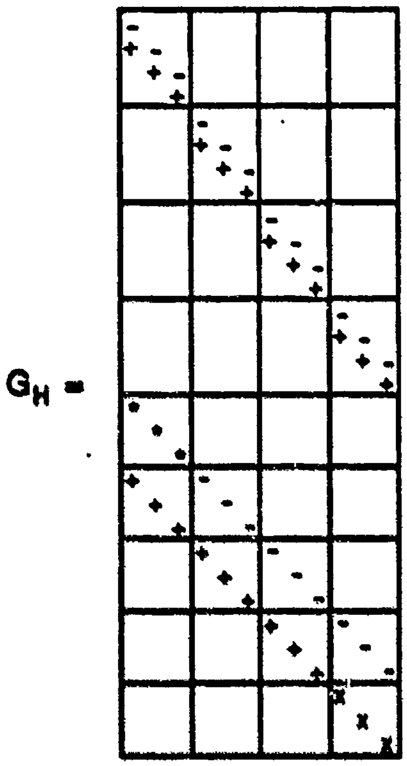
$$E_0 = \begin{bmatrix} E \\ e^T \end{bmatrix} \quad \text{and} \quad \mu = \begin{bmatrix} \hat{\mu} \\ 0 \end{bmatrix}. \tag{2.29}$$

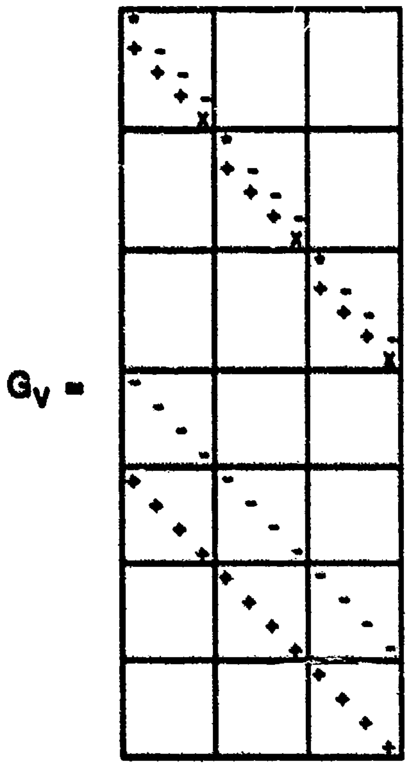
Since $E_0^T\mu = E^T\hat{\mu}$, replacing E_0 with E (and μ with $\hat{\mu}$) is equivalent to "grounding" a single pressure node to zero. Thus we can safely replace E_0 with E throughout the saddle point system (2.28). The matrix E has full row rank, so hypothesis H1 holds. Morever, F is symmetric positive definite, so there exists a G of full column rank such that $F = G^TG$. Then, since G has no non-trivial nullspace, hypothesis H2 holds vacuously.

There is still one important task remaining. We will need the matrix G to construct one type of preconditioner for the Stokes problem (see chapter 5). The Cholesky factor of F, however, is far too dense to be practical for this purpose. Instead, we exploit the fact that the Laplacian operator is the divergence of the gradient. We can therefore define G^T to be the appropriate discretization of the divergence acting on the velocity edges of the grid. Since the adjoint of the divergence is the negative of the gradient, we find that G satisfies $F = G^TG$ as required. Using centered differences and techniques described in Strang [35], we obtain G of the form

$$G = \frac{1}{\sqrt{h_s}} \begin{bmatrix} G_H & \\ & G_V \end{bmatrix}, \tag{2.30}$$

where $-G_H$ and $-G_V$ are discrete gradients acting on the horizontal and vertical edges of the grid respectively. Figure 2.10 depicts the matrices for the case n_s=4;





"+"	=	1
"-"	=	-1
"x"	=	$\sqrt{2}$
"*"	=	$-\sqrt{2}$

Figure 2.10: Blocks of G for the Stokes Problem

the pattern is the same for larger n_s as well. It will prove helpful to observe that the rows of G can be reordered so that the upper block of the matrix is upper triangular and non-singular.

2.4 Substructured Problems

When working with discrete models such as those in §2.2 and §2.3, the ordering of the nodes and elements is an important issue. The choice of ordering affects virtually every aspect of computational performance, including program complexity, storage requirements, and convergence of iterative schemes. Here we discuss a standard way to order the nodes and elements to improve the opportunities for parallel computation. The ordering technique, known as **substructuring** or **domain decomposition**, involves grouping the components of the model into contiguous pieces. The literature on the subject is vast; see Ortega and Voigt [27] for an annotated introduction.

Consider, for example, the structure WRENCH depicted in 2.11 (this model is one of six small test problems developed by M. Lawo [6]). Partition the model into a desired number of disjoint **substructures** such that each node is in exactly one substructure. There are now two types of elements in the model. Most elements (the dark shaded elements in the figure) interact only with nodes in a single substructure. Other elements, called **transition elements** (the lightly shaded elements in the figure), interact with nodes from more than one substructure. Number the nodes and elements in the logical way: all nodes (and elements) in substructure A, then B, etc. Defer the transition elements until last, regardless of their physical location in the structure. For our purposes, it is neither necessary nor appropriate to use transition *nodes* in substructuring the

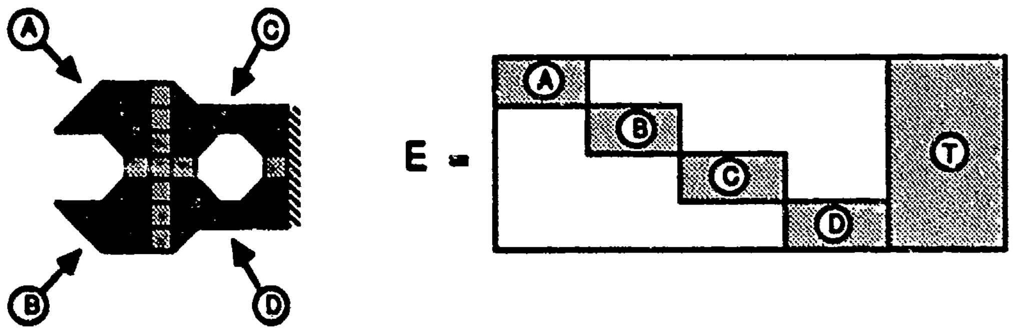


Figure 2.11: Substructuring a Finite Element Model

model; see Plemmons and White [31] for a description of the matrices resulting from such a substructuring.

Because the interactions between the nodes and the elements determine the non-zeros in the equilibrium matrix, we find that the partitioned model leads to a matrix E with non-zeros confined to the blocks shown in the figure. Elements in substructure A, for example, produce non-zeros confined to the "diagonal" block labelled A, while the transition elements produce the transition block labelled T in the figure. The matrices F and G, which are block diagonal with one small block for each element, retain their structure regardless of the ordering.

The diagonal blocks in the substructured equilibrium matrix deserve further comment. Note that each diagonal block of E is itself an equilibrium matrix for the associated portion of the model viewed as an independent structure. If this physical substructure is stable (see §2.2), then the diagonal block has full row rank; otherwise, it is rank deficient. In the example, substructures C and D are stable (they are fixed to the immovable base on the right side, and have sufficient internal support), while substructures A and B are not (they have no

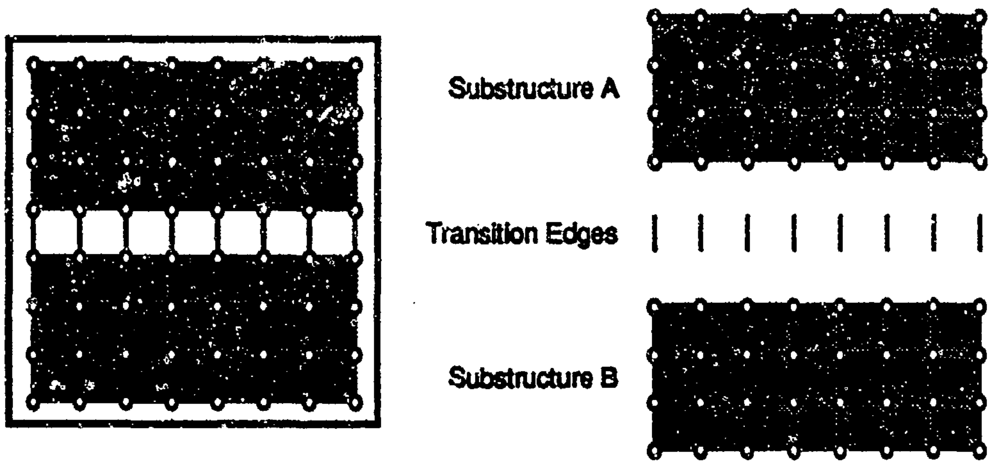


Figure 2.12: Narrow Substructuring of MAC Grid

external support). Thus the first two diagonal blocks in E lack full row rank.

Later, we address ways to exploit the substructured form of the matrix E. For now, we emphasize that the ideal situation is one in which the diagonal blocks are all roughly the same size, the number of substructures is the same as the number of available processors, and the transition block has relatively few columns. Problem WRENCH is far too small to justify the use of four substructures in practice; we use it only to illustrate the concept.

The process of substructuring the MAC grid for the Stokes problem is similar to what we have just done, but is somewhat more delicate. Begin with the grid derived in the previous section, and partition the pressure nodes into substructures (see the left portion of figure 2.12 for an example involving two substructures; for clarity, we do not include arrows on the velocity edges). Let the vertical edges connecting the substructures serve as the transition elements as shown in the exploded view (right half of the figure). This leads to a substructured equilibrium matrix analogous to that derived above, just as one would expect.

This substructuring of the MAC grid is useful for some purposes; we employ it for one set of experiments in chapter 6. It is deficient in one important respect, however: for this partitioning of the edges, the matrices F and G do not reflect the substructuring. To see why, first observe that E is an node-edge matrix in the sense that the rows of E correspond to nodes in the grid, while the columns of E correspond to edges (see Strang [35]). Thus, a substructured matrix E results when *edges* in a given substructure interact only with *nodes* in that substructure. The matrix F, on the other hand, is an edge-edge matrix: both the rows and the columns of the matrix are associated with edges in the grid. Thus, we obtain a substructured form for F when *edges* in a given substructure interact only with other *edges* in that substructure. But the Laplacian stencil defining the non-zeros due to the horizontal edges (§2.3) causes a problem. Consider a stencil centered at a horizontal edge on the top boundary of the lower substructure (figure 2.13; the edges involved in the stencil appear as thick lines in the magnified view). Note that the stencil involves edges from both the upper and lower substructures, violating the requirement that the two substructures do not interact (the vertical stencil causes no problem in this example).

We can solve this difficulty with a simple change. We leave the partitioning of the nodes unchanged, but include as transition edges those horizontal edges which are adjacent to a transition zone. Thus, a given transition zone in the physical grid consists of a row of vertical edges and two rows of horizontal edges, as shown in the left portion of figure 2.14. With this choice of transition edges, the symmetric matrix F has the structure shown in the right half of the figure. There is a large diagonal block associated with each substructure; in

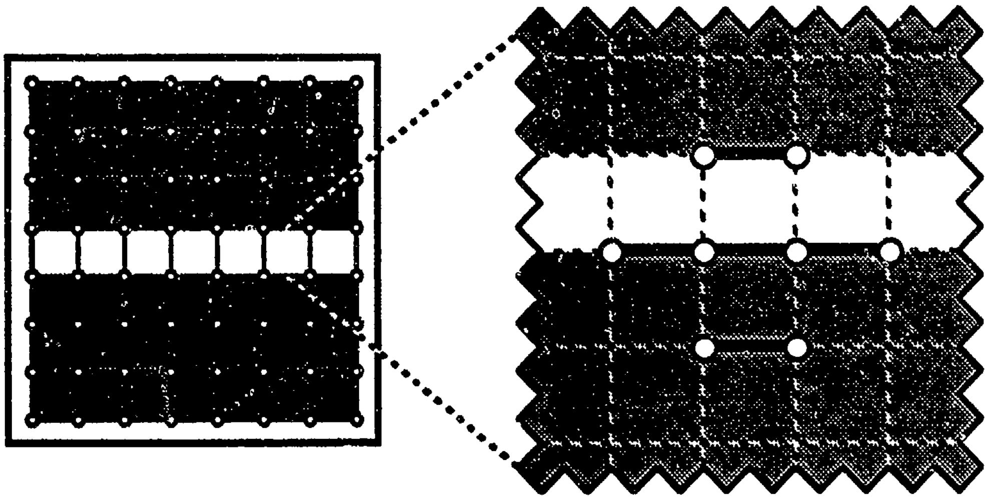


Figure 2.13: MAC Stencil (Horizontal Momentum) at a Transition Line

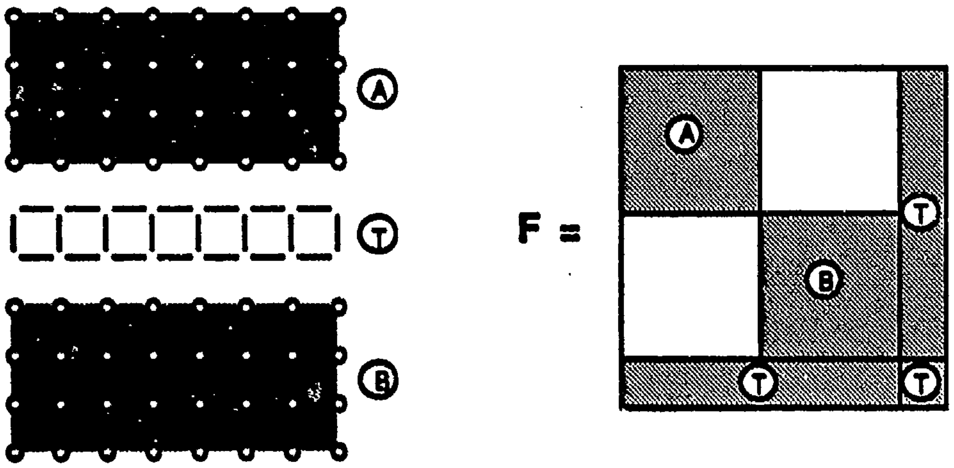


Figure 2.14: Wide Substructuring of MAC Grid

fact, each such block is the Laplacian acting on the substructure, and has the form described in equations (2.25) and (2.26). Additionally, there are horizontal and vertical strips associated with transition edges. The matrix G, which comes from a gradient stencil that is a subset of the Laplacian stencil used for F (recall §2.3), has a substructured form resembling the transpose of the equilibrium matrix. We will make use of the substructured form of both F and G in one of the algorithms we develop in chapter 5.

3. Description of the Algorithms

In this chapter, we introduce four iterative algorithms for solving problem LSE: p-cyclic successive-overrelaxation (SOR), block accelerated overrelaxation (AOR), a preconditioned form of the weighting method, and of course algorithm BNP. We compare and contrast these algorithms in chapter 4.

Three of these algorithms (p-cyclic SOR, block AOR, and algorithm BNP) iterate on a certain modified form of the Kuhn-Tucker equations; the fourth (preconditioned weighting) is closely related to this modified form. We therefore begin by constructing the **modified Kuhn-Tucker equations** (§3.1), and include a discussion of the computations necessary to achieve this form. We emphasize the parallel aspects of the computation, describing a technique called **interlacing** proposed by James and Plemmons [20]. We outline each of the four algorithms in the remaining sections. In §3.2 we detail algorithm BNP as introduced in [4]. We mention a source of inaccuracy in the original version of the algorithm (discussed in more detail in chapters 5 and 6), and propose a simple correction. Section 3.3 is an introduction to the other three iterative algorithms; we emphasize established results which will prove important in subsequent chapters.

3.1 Modified Kuhn-Tucker Equations

It is difficult to apply traditional iterative methods to the Kuhn-Tucker equations as written in (2.6): the diagonal blocks are not even square, let alone non-singular. We therefore start by repartitioning these equations. Since we are assuming E has full row rank, and that $\begin{bmatrix} E \\ G \end{bmatrix}$ has full column rank, we can reorder the rows of G and c, and repartition

$$G = \begin{bmatrix} G_1 \\ G_2 \end{bmatrix} \tag{3.1}$$

so that the matrix defined by

$$A_1 = \begin{bmatrix} E \\ G_1 \end{bmatrix} \tag{3.2}$$

is square and non-singular. Defining $A_2 = G_2$ for convenience, we now have

$$\begin{bmatrix} E \\ G \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}. \tag{3.3}$$

Make these substitutions in (2.6) and reorder the column blocks to obtain the **modified Kuhn-Tucker equations:**

$$\begin{bmatrix} A_1 & 0 & K \\ A_2 & I & 0 \\ 0 & A_2^T & A_1^T \end{bmatrix} \begin{bmatrix} y \\ r_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ c_2 \\ 0 \end{bmatrix}, \tag{3.4}$$

where $c = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$, $r = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$, $b_1 = \begin{bmatrix} b \\ c_1 \end{bmatrix}$, $x_3 = \begin{bmatrix} \lambda \\ r_1 \end{bmatrix}$, and $K = \begin{bmatrix} 0 & 0 \\ 0 & I \end{bmatrix}$. Note that the diagonal blocks are now square and non-singular.

We also observe that the ordinary (unconstrained) least squares problem

$$\text{minimize } \|Gy - c\|_2 \tag{3.5}$$

may be viewed as a "constrained" problem with zero constraints. We can reorder the rows of G and c as necessary, and partition $G = \begin{bmatrix} G_1 \\ G_2 \end{bmatrix}$ so that G_1

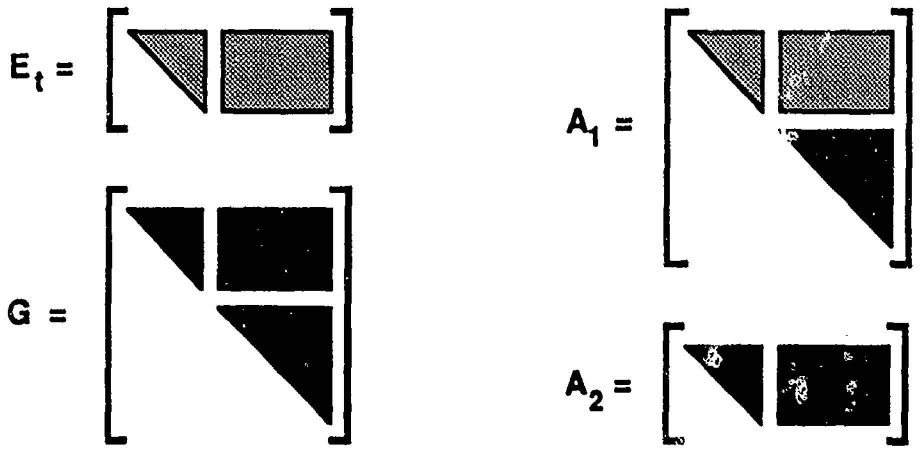


Figure 3.1: Forming A_1 from trapezoidal E_t

is square and non-singular. It then becomes easy to write a set of modified Kuhn-Tucker equations analogous to equation (3.4): A_1 consists entirely of G_1, b_1 equals c_1, and an identity matrix I replaces K. In chapter 4, it will prove helpful to view the unconstrained problem this way.

In each of the algorithms we consider, we will repeatedly solve systems of equations which have A_1 and A_1^T as the coefficient matrix. Thus, the key to using the modified Kuhn-Tucker equations is producing an A_1 which is easily invertible. The most convenient form to seek is an upper triangular matrix. If G has full column rank (a stronger assumption than H2), one way to accomplish this, depicted in figure 3.1, is as follows:

1. Use Gauss Elimination (or orthogonal reduction) with column pivoting to replace E with the upper trapezoidal matrix

$$E_t = \left[\begin{array}{cc} E_L & E_R \end{array} \right], \tag{3.6}$$

where E_L is upper triangular and non-singular. Here the subscript t indicates "trapezoidal," while L and R indicate left and right respectively.

2. Apply the same column interchanges to G, and partition compatibly with E_t. Use orthogonal rotations on G and c to replace G with the non-singular upper triangular matrix

$$G = \begin{bmatrix} G_{21} & G_{22} \\ & G_{12} \end{bmatrix}. \tag{3.7}$$

Here the leading subscripts indicate whether the blocks will become part of G_1 or G_2.

3. Define $G_1 = \begin{bmatrix} 0 & G_{12} \end{bmatrix}$ and $G_2 = \begin{bmatrix} G_{21} & G_{22} \end{bmatrix}$, giving us

$$A_1 = \begin{bmatrix} E_L & E_R \\ & G_{12} \end{bmatrix}. \tag{3.8}$$

While this method of constructing A_1 is useful for analysis, it may not be efficient: column interchanges may destroy exploitable structure in the matrices E and G. Another approach, first proposed by James and Plemmons [20], is to reduce E without column pivoting to produce a "stairstep" form E_s (figure 3.2). Now form A_1 by interlacing rows of G with rows of E_s. A permutation matrix

$$P = \begin{bmatrix} P_L & P_R \end{bmatrix} \tag{3.9}$$

defines the interlacing:

$$A_1 = P \begin{bmatrix} E_s \\ G_1 \end{bmatrix} = P_L E_s + P_R G_1. \tag{3.10}$$

Notice that the same permutation matrix (not its transpose) relates the stairstep and trapezoidal forms:

$$E_s P = \begin{bmatrix} E_s P_L & E_s P_R \end{bmatrix} = \begin{bmatrix} E_L & E_R \end{bmatrix} = E_t. \tag{3.11}$$

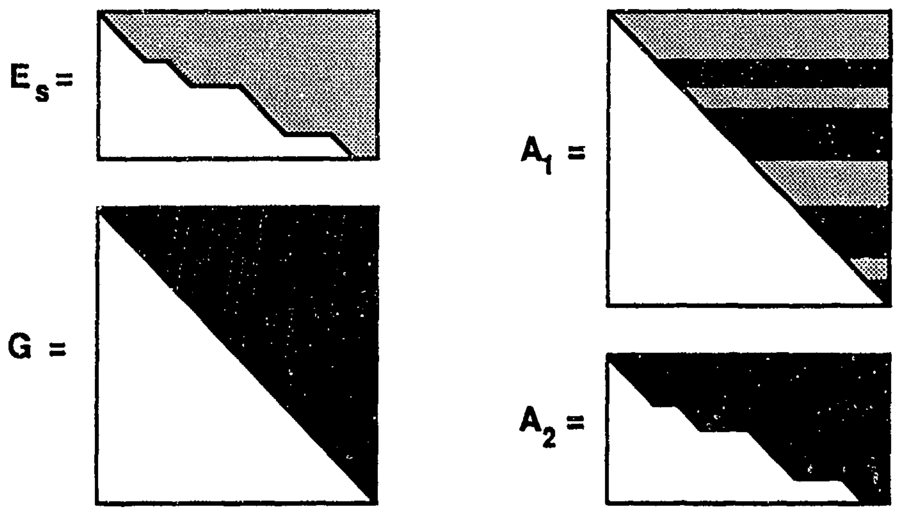


Figure 3.2: Forming A_1 from stairstep E_s

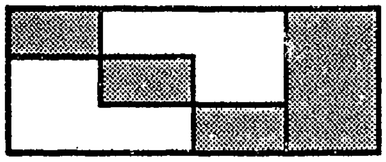
When the matrices E and G reflect a substructuring of the physical domain, interlacing without column permutations is especially helpful for parallel computations. Given a substructured equilibrium matrix, we can produce the stairstep matrix E_s in parallel by assigning each diagonal block of E to a separate processor. The only subtle issue is the possibility that a given diagonal block does not have full row rank. To account for this possibility, we produce the matrix E_s as follows (figure 3.3):

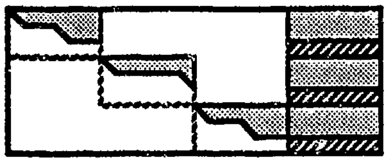
1. Begin with E in substructured form.

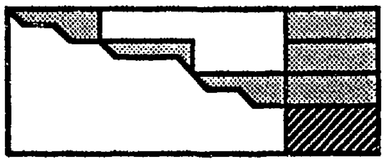
2. Factor each row block in parallel. Terminate computations on a row when the leading non-zero in that row is in the transition block.

3. Consider all rows which now have leading non-zeros in the transition block. Move them (implicitly, using a pointer vector) to the bottom of the matrix.

4. Factor the transition block.


a. Begin with Substructured E


b. Factor Row Blocks in Parallel


c. Move "Transition Rows" to Bottom

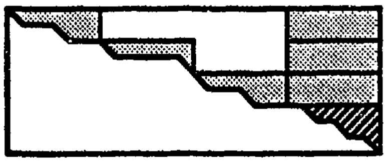

d. Factor "Transition Rows"

Figure 3.3: Forming Stairstep E_s from Substructured E

If column pivoting is needed or desired when reducing E, one can preserve the structure of E by restricting eligible pivot columns to those associated with the current diagonal block of the matrix (however one must consider the effect of such pivoting on both the matrix F and the intermediate fill required to generate G). In any case, one can now complete the construction of A_1 by interlacing (figure 3.4) exactly as we did above.

Notice that triangular solves involving A_1 provide opportunities for block-based parallelism. One first solves for the unknowns associated with columns of the transition block (i.e. the final block of unknowns). Once this is complete, one can recover the remaining unknowns in parallel by assigning each diagonal block of A_1 to a separate processor. A similar algorithm applies to triangular solves with the matrix A_1^T as well.

One triangular solve of each type will occur in each iteration of the algo-

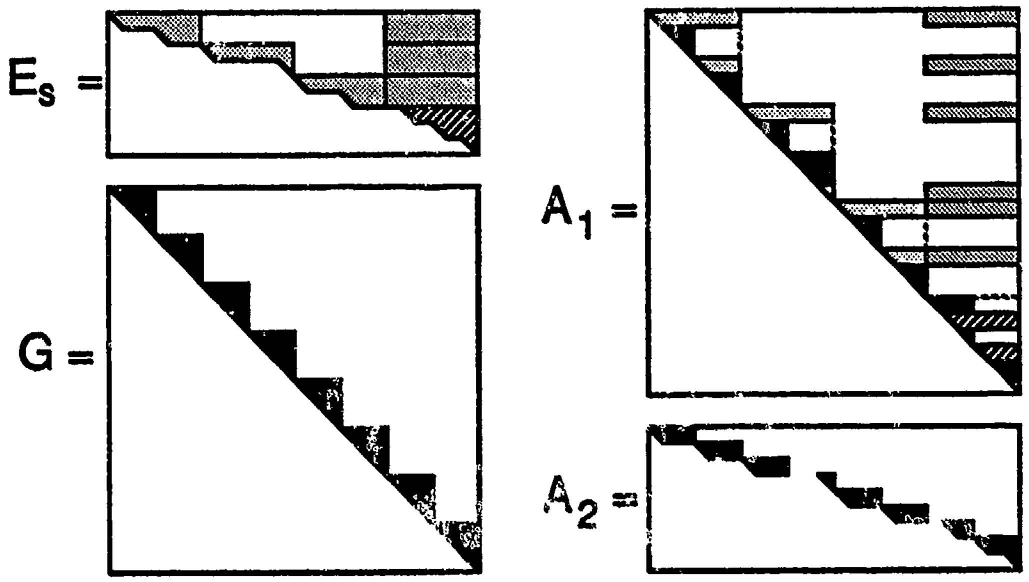


Figure 3.4: Forming A_1 from Substructured E_s

rithms we consider; in fact, these solves will account for a major percentage of the execution time. Clearly the size of the transition block is a key limiting factor in the parallel performance of these computations. Also important, however, are the relative size and structure of the diagonal blocks of A_1: if these blocks are of widely varying size or sparsity, the parallel performance of the triangular solves will suffer.

3.2 Algorithm BNP

The derivation of Algorithm BNP as presented in Barlow, Nichols, and Plemmons [4] begins with the modified Kuhn-Tucker equations

$$\begin{bmatrix} A_1 & 0 & K \\ A_2 & I & 0 \\ 0 & A_2^T & A_1^T \end{bmatrix} \begin{bmatrix} y \\ r_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ c_2 \\ 0 \end{bmatrix}, \tag{3.12}$$

where the symbols are defined in equation (3.4). The authors then rewrite this system, applying block elimination to reduce it to block upper triangular form. The result of this reduction is the system

$$\begin{bmatrix} A_1 & 0 & K \\ & I & -A_2A_1^{-1}K \\ & & (A_1^T + A_2^TA_2A_1^{-1}K) \end{bmatrix} \begin{bmatrix} y \\ r_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ c_2 - A_2A_1^{-1}b_1 \\ A_2^T(A_2A_1^{-1}b_1 - c_2) \end{bmatrix}. \tag{3.13}$$

Continuing as in [4], multiply the third block equation by A_1^{-T} to obtain an unsymmetric system in the unknown z_3:

$$(I + A_1^{-T}A_2^TA_2A_1^{-1}K)z_3 = A_1^{-T}A_2^T(A_2A_1^{-1}b_1 - c_2). \tag{3.14}$$

To simplify this system, temporarily define $B = A_1^{-T}A_2^TA_2A_1^{-1}$, noting that B is symmetric non-negative definite. Then, in terms of B, the system in z_3 has the form

$$(I + BK)z_3 = d, \tag{3.15}$$

where d temporarily represents the righthand side in (3.14).

Let

$$\hat{K} = \begin{bmatrix} 0 \\ I \end{bmatrix} \tag{3.16}$$

be the rightmost block of the matrix K, noting that $\hat{K}^T\hat{K} = I$ and $\hat{K}\hat{K}^T = K$. Moreover, observe that an arbitrary matrix-matrix product of the form $S\hat{K}$ is simply the right block of S, while $\hat{K}^TS$ produces the lower block of S.

Now partition B compatibly with K and $\hat{K}$:

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{12}^T & B_{22} \end{bmatrix}. \tag{3.17}$$

Given this partitioning, equation (3.14) can be written as

$$\begin{bmatrix} I & B_{12} \\ 0 & (I + B_{22}) \end{bmatrix} \begin{bmatrix} \lambda \\ r_1 \end{bmatrix} = d. \tag{3.18}$$

Thus the lower block in the block upper triangular system (3.13) is itself block upper triangular, and from its lowest block we obtain an equation in the unknown r_1:

$$(I + B_{22})r_1 = \bar{K}^T d. \qquad (3.19)$$

B_{22} is symmetric non-negative definite, so $(I + B_{22})$ is symmetric positive definite. If we define the matrix

$$Y = A_2 A_1^{-1} \bar{K}, \qquad (3.20)$$

then B_{22} is simply $Y^T Y$, and the system in r_1 is the symmetric positive definite system

$$(I + Y^T Y)r_1 = h, \text{ where } h = Y^T(A_2 A_1^{-1} b_1 - c_2). \qquad (3.21)$$

Algorithm BNP solves this system by the conjugate gradient algorithm, working with the coefficient matrix in factored form. For this reason, we will often refer to (3.21) simply as the BNP system. There are $n - m_1$ unknowns in the system, providing a theoretical upper bound on the maximum number of conjugate gradient iterations required for convergence. Moreover, as Barlow, Nichols, and Plemmons observe in [4], Y may lack full column rank:

$$\begin{aligned} \text{Rank}(Y) &\le \text{Rank}(\bar{K}) \\ &= n - m_1 \qquad (3.22) \\ \text{Rank}(Y) &\le \text{Rank}(A_2) \\ &\le \min(n, m_1 + m_2 - n). \qquad (3.23) \end{aligned}$$

This gives us

$$\begin{aligned} \text{Rank}(Y) &\le \min(n - m_1, m_1 + m_2 - n) \\ &= \min(\text{ \# of rows in } G_1, \text{ \# of rows in } G_2 \text{ }). \qquad (3.24) \end{aligned}$$

If Y does in fact lack full column rank, then zero is an eigenvalue of Y^TY (with multiplicity determined by the rank deficiency), and the coefficient matrix $(I+Y^TY)$ has clustered eigenvalues at unity. This further reduces the theoretical maximum number of conjugate gradient iteratio: s required for convergence.

If G is square (as it is, for example, in the engineering application), then $m_2=n$, and the clustering analysis simplifies somewhat. For $m_1 \leq \frac{1}{2}n-1$, the maximum number of iterations is controlled by the column rank of Y, and must be less than or equal to m_1+1, which in turn is no larger than $\frac{1}{2}n$. For $m_1 \geq \frac{1}{2}n$, the maximum number of iterations is bounded by the size of the BNP system, and must be less than or equal to $n-m_1$, which in turn is no greater than $\frac{1}{2}n$. Thus the theoretical upper bound on the iterations is at most $\frac{1}{2}n$, with the worst case occurring when $m_1 = \frac{1}{2}n$. (At first glance, there is an apparent paradox here: given an ordinary least squares problem, which has zero constraints, the analysis above suggests we should expect convergence in one iteration. But if we have an *ordinary* least squares problem with a *square* G, then $A_1 = G$. We are presuming that systems which have A_1 as the coefficient matrix are easily solved. If this were the case for an ordinary least squares problem, we would have a trivial problem, and there would be no need to iterate at all.)

Of course, our goal is to find y, not just r_1. In principle, we could iterate until convergence to solve for r_1, then backsolve through equations (3.15) and (3.13) to recover the solution y. In practice, however, we have another alternative: the analysis in [4] shows that a variation of the conjugate gradient algorithm applied to the BNP system generates all that is necessary to obtain an estimate for y at each iteration.

To see why this is so, first note that $Kz_3 = \bar{K}r_1$. This allows us to rewrite the first equation in the modified Kuhn-Tucker system (3.12), producing a formula relating y and r_1:

$$A_1 y + \bar{K} r_1 = b_1. \tag{3.25}$$

Now apply the conjugate gradient algorithm to the BNP system, using an arbitrary guess of $r_1^{(0)}$. Let $r_1^{(k)}$ represent the approximation to r_1 at the kth iterate, and use equation (3.25) to *define* a corresponding approximation to y:

$$y^{(k)} = A_1^{-1}(b_1 - \bar{K} r_1^{(k)}). \tag{3.26}$$

Initialize $y^{(0)}$ consistent with this definition.

At each iteration of the conjugate gradient algorithm, we will correct the estimate of r_1 in the standard way, using a recursion of the form

$$r_1^{(k+1)} = r_1^{(k)} - \gamma_k s_k. \tag{3.27}$$

Here s_k is a direction vector, and γ_k scales that vector to the appropriate length. Now from equation (3.26), we know that $y^{(k+1)} = A_1^{-1}(b_1 - \bar{K} r_1^{(k+1)})$. Substitute (3.27) into the latter expression, and simplify to obtain

$$y^{(k+1)} = y^{(k)} + \gamma_k q_k, \quad \text{where} \quad q_k = A_1^{-1} \bar{K} s_k. \tag{3.28}$$

Thus we can update $y^{(k)}$ by a multiple of the direction vector q_k in a manner typical of the conjugate gradient method. The scale factor γ_k is already available; we need it to update $r_1^{(k)}$. The vector q_k also occurs naturally in the conjugate gradient iteration, so we can produce $y^{(k)}$ at each iteration for a fairly modest cost (one so-called dense vector triad per iteration). Depending on the number of iterations required to achieve convergence, this cost may or may not be less than the cost of the single $n \times n$ sparse triangular solve needed to determine y once the iterations are complete. In any case, the difference in the two approaches is not likely to be a significant portion of the overall computational effort, and both methods have performed with comparable accuracy and speed in our experiments.

There is one other aspect of Algorithm BNP which differs from the traditional conjugate gradient algorithm. Let

$$\nu_k = (I + Y^T Y) r_1^{(k)} - \zeta \tag{3.29}$$

represent the defect associated with the BNP system (we use the term defect to avoid confusion with the least squares residual r). Recall that the quantity $\nu_k^T \nu_k$ is needed within a conjugate gradient iteration to compute various scale factors; it is also used to test for convergence. If we were to apply the classical conjugate gradient algorithm to the BNP system, we would compute the defect using an update of the form

$$\nu_{k+1} = \nu_k + \alpha_k \hat{s}_k, \tag{3.30}$$

where $\hat{s}_k = (I + Y^T Y) s_k$ is a direction vector, α_k is a scale factor, and both these quantities are needed elsewhere in the iteration. Barlow, Nichols, and Plemmons take a different approach, however. Define

$$r_2^{(k)} = c_2 - A_2 y^{(k)} \tag{3.31}$$

to be the kth approximation to r_2. If we initialize $r_2^{(0)}$ to the value required by its definition, we can show by an argument similar to the one above that the iterates satisfy the recursion

$$r_2^{(k+1)} = r_2^{(k)} - \gamma_k Y s_k. \tag{3.32}$$

We can now use $r_2^{(k)}$ to compute the defect. Begin by substituting equation (3.26) into (3.29) to find that

$$\nu_k = r_1^{(k)} + Y^T(c_2 - A_2 y^{(k)}). \tag{3.33}$$

Then use the definition of $r_2^{(k)}$ given in (3.31) to obtain

$$\nu_k = r_1^{(k)} + Y^T r_2^{(k)}. \tag{3.34}$$

This method of computing the defect requires the same computational effort as the standard method of explicitly using a scale factor and direction vector.

We summarize the complete algorithm in table 3.1. This particular outline obscures opportunities to avoid redundant calculations, but will prove useful for the analysis in chapter 4. See the original paper [4] for a version suitable for implementation.

Our experiments suggest, however, that equation (3.34) is somewhat unstable: we obtained superior results with a conventional update of the defect, especially on large problems (the BNP method of updating the vector y appears to present no difficulties). Apparently the defect gradually drifts from the correct value, causing the direction vectors and scale factors to become inaccurate, and resulting in slower convergence. Barlow, in a private communication [3], has observed that this is most likely because the independent updates of r_1, r_2, and y do not enforce the relationships which must exist among these three quantities. Fortunately, the problem is quite easy to correct (see chapters 5 and 6): a traditional update of the defect improves performance, while preserving the spirit of the BNP algorithm.

We remark in passing that algorithm BNP has one other interesting property. The quantity $r_2^{(k)}$ is *defined* to be equal to $c_2 - G_2 y^{(k)}$. Additionally, a

Table 3.1: Algorithm BNP (Problem LSE)

```
1. Initialize:
    (a) $y^{(0)}$ arbitrary    (normally $y^{(0)} = A_1^{-1}b_1$)
    (b) $r_1^{(0)} = \bar{K}^T(b_1 - A_1 y^{(0)}) = c_1 - G_1 y^{(0)}$    (normally $r_1^{(0)} = 0$)
    (c) $r_2^{(0)} = c_2 - A_2 y^{(0)} = c_2 - G_2 y^{(0)}$
    (d) $\nu_0 = r_1^{(0)} + Y^T r_2^{(0)}$    ($\nu_k$ is the defect $(I + Y^T Y) r_1^{(k)} - h$)
    (e) $s_0 = \nu_0$    ($s_k$ is the direction vector)
2. For $k = 0, 1, \ldots,$ until $\nu_k^T \nu_k <$ tolerance:
    (a) $\gamma_k = \nu_k^T \nu_k / s_k^T (I + Y^T Y) s_k$
    (b) $r_1^{(k+1)} = r_1^{(k)} - \gamma_k s_k$
    (c) $y^{(k+1)} = y^{(k)} + \gamma_k A_1^{-1} \bar{K} s_k$
    (d) $r_2^{(k+1)} = r_2^{(k)} - \gamma_k Y s_k$
    (e) $\nu_{k+1} = r_1^{(k+1)} + Y^T r_2^{(k+1)}$
    (f) $\beta_{k+1} = \nu_{k+1}^T \nu_{k+1} / \nu_k^T \nu_k$
    (g) $s_{k+1} = \nu_{k+1} + \beta_{k+1} s_k$
```

simple induction argument confirms that the vector $r_1^{(k)}$, the BNP unknown, satisfies a similar natural relationship: $r_1^{(k)} = c_1 - G_1 y^{(k)}$. This means that the vector $\begin{bmatrix} r_1^{(k)} \\ r_2^{(k)} \end{bmatrix}$ is precisely the residual $r^{(k)} = c - Gy^{(k)}$ associated with the kth approximation to the solution vector y. Of the four basic iterative algorithms we will consider, algorithm BNP is the only one with this satisfying property.

3.3 Other Iterative Algorithms

There are, of course, other established iterative algorithms for solving problem LSE. Two such methods, p-cyclic SOR and block AOR, are parameter-based linear stationary methods applied to the modified Kuhn-Tucker equations (3.12). The third, a preconditioned form of the weighting algorithm, is closely related to the modified Kuhn-Tucker formulation. We introduce all three algorithms here, with an eye toward comparing and contrasting these methods with BNP in chapter 4.

Given a non-singular linear system $Cz = f$, let $C = D - L - U$ define a splitting of the coefficient matrix into block diagonal, lower triangular, and upper triangular parts respectively. The well-known block SOR algorithm (see Young [38], and Hageman and Young [14]) is then defined by an iteration involving a relaxation parameter ω:

$$(D - \omega L)z^{(k+1)} = [(1-\omega)D + \omega U]\, z^{(k)} + \omega f. \tag{3.35}$$

If we consider the modified Kuhn-Tucker system (3.12), two plausible choices of the block diagonal matrix D produce three- and two-block SOR methods respectively:

$$D_3 = \begin{bmatrix} A_1 & & \\ & I & \\ & & A_1^T \end{bmatrix} \quad \text{and} \quad D_2 = \begin{bmatrix} A_1 & & \\ A_2 & I & \\ & & A_1^T \end{bmatrix}. \tag{3.36}$$

While little is known about the optimal iteration parameter ω for arbitrary linear systems, the coefficient matrix in the modified Kuhn-Tucker system enjoys some special properties which make a more complete analysis possible. When the coefficient matrix of the modified Kuhn-Tucker system is partitioned using either D_3 or D_2, it is a so-called **p-cyclic matrix**, and the associated SOR algorithms are known as **p-cyclic SOR methods**. An elegant theory, due largely to Young [39] and Varga [36], [37], relates the spectrum of the SOR iteration matrix to that of the corresponding Jacobi iteration matrix. This p-cyclic theory, combined with special properties of the Jacobi iteration matrices associated with D_2 and D_3, leads to a number of important results for p-cyclic SOR applied to problem LSE. In particular, Plemmons [30] has established that 2-cyclic SOR applied to LSE converges for sufficiently small values of ω (there may or may not be values of ω for which 3-cyclic SOR converges). Additionally, Markham, Neumann, and Plemmons [24] have shown that the asymptotic convergence of optimal 2-cyclic SOR is superior to 3-cyclic SOR. Pierce, Hadjidimos, and Plemmons [29], and Eiermann, Niethammer, and Ruttan [10], establish results for more general problems, demonstrating the importance of the special properties of the modified Kuhn-Tucker system.

Another approach to solving the modified Kuhn-Tucker equations is a two parameter generalization of SOR known as **Accelerated Over-relaxation** or **AOR** (see, for example, Hadjidimos [13]):

$$(D - \beta L)z^{(k+1)} = [(1-\omega)D + (\omega - \beta)L + \omega U]\, z^{(k)} + \omega f. \tag{3.37}$$

Note that the special case $\omega = \beta$ is block SOR.

Again we consider the blockings given by (3.36), referring to the corresponding algorithms as 3-AOR and 2-AOR respectively. The optimal choice of parameters for 2-AOR occurs somewhere on the line $\omega = \beta$ (see Papadopoulou, Saridakis, and Papatheodorou [28]), so optimal 2-AOR coincides with optimal 2-cyclic SOR. Optimal 3-AOR, however, does not necessarily occur when $\omega = \beta$. In fact, Papadopoulou *et. al.* establish in [28] that under some very restrictive (and highly technical) conditions, the asymptotic convergence of 3-AOR may be better than optimal 2-AOR (and therefore 2-cyclic SOR).

Table 3.2 is an explicit description of the 3-AOR algorithm applied to problem LSE. Remember that the matrices K and $\bar{K}$ are defined in equations (3.12) and (3.21). Technically, we could initialize $r_1^{(0)}$ and $r_2^{(0)}$ to arbitrary values. As defined in the table, however, the initial values are both mathematically plausible and consistent with the BNP iteration (unlike BNP, however, subsequent iterates do not satisfy $r_1^{(k)} = c_1 - G_1 y^{(k)}$ or $r_2^{(k)} = c_2 - A_2 y^{(k)}$). In any case, our experiments suggest the method is not particularly sensitive to the choice of initial iterates.

We can solve problem LSE in yet another way, this time by viewing the problem in an entirely different manner. Let τ be a large positive constant, and consider the ordinary least squares problem

$$\text{minimize } \|Wy - \hat{b}\|_2, \tag{3.38}$$

where W and $\hat{b}$ are given by

$$W = \begin{bmatrix} \tau E \\ G \end{bmatrix}, \quad \hat{b} = \begin{bmatrix} \tau b \\ c \end{bmatrix}. \tag{3.39}$$

Table 3.2: Three-Block AOR (Problem LSE)

```
1. Initialize:
    (a) $y^{(0)}$ arbitrary
    (b) $r_2^{(0)} = c_2 - A_2 y^{(0)}$
    (c) $r_1^{(0)} = c_1 - G_1 y^{(0)}$
    (d) $\lambda^{(0)}$ arbitrary
    (e) $z_3^{(0)} = \begin{bmatrix} \lambda^{(0)} \\ r_1^{(0)} \end{bmatrix}$
2. For $k = 0, 1, \ldots,$ until convergence:
    (a) $y^{(k+\frac{1}{2})} = A_1^{-1}(b_1 - K z_3^{(k)}) = A_1^{-1}(b_1 - \tilde{K} r_1^{(k)})$
    (b) $r_2^{(k+\frac{1}{2})} = c_2 - A_2 \left[(1-\beta) y^{(k)} + \beta y^{(k+\frac{1}{2})}\right]$
    (c) $z_3^{(k+\frac{1}{2})} = -A_1^{-T} A_2^T \left[(1-\beta) r_2^{(k)} + \beta r_2^{(k+\frac{1}{2})}\right]$
    (d) $y^{(k+1)} = (1-\omega) y^{(k)} + \omega y^{(k+\frac{1}{2})}$
    (e) $r_2^{(k+1)} = (1-\omega) r_2^{(k)} + \omega r_2^{(k+\frac{1}{2})}$
    (f) $z_3^{(k+1)} = (1-\omega) z_3^{(k)} + \omega z_3^{(k+\frac{1}{2})}$
```

This is a special case of a so-called **weighted least squares problem**, or **WLS** (see, for example, Bjorck [7], or Golub and van Loan [12]).

Note that

$$\|Wy - \hat{b}\|_2^2 = \tau^2\|Ey - b\|_2^2 + \|Gy - c\|_2^2. \tag{3.40}$$

This means that any candidate solution y which fails to satisfy $Ey \approx b$ will cause the quantity $\|Wy - \hat{b}\|_2^2$ to be exceptionally large. Thus, τ acts as a penalty parameter forcing y to come very close to satisfying the equilibrium constraint, and we might expect that the solution to this weighted least squares problem is, to a good approximation, the solution to problem LSE. Given a reasonably conditioned problem and a well chosen value of τ, the analysis in Barlow [2] suggests that this is in fact the case.

This approach to solving problem LSE, known as the **weighting method**, has a fairly long history (see Bjorck [7]). Most algorithms based on this method solve the weighted least squares problem (3.38) by some direct method (e.g. an orthogonal factorization). Here, however, we consider an iterative approach based on the factored form of the normal equations

$$W^TWy = W^T\hat{b}. \tag{3.41}$$

Anticipating the likelihood that this system is poorly conditioned, we seek to precondition the problem.

To construct a preconditioner, first note that the matrix W as defined above is nothing more than a scaled version of the matrix $\begin{bmatrix} E \\ G \end{bmatrix}$ which appears in the Kuhn-Tucker equations (2.6). Thus, we can reorder and repartition it exactly as in §3.1. In particular, reorder the rows of G and c, and partition

$$G = \begin{bmatrix} G_1 \\ G_2 \end{bmatrix} \tag{3.42}$$

so that the matrix defined by

$$W_1 = \begin{bmatrix} \tau E \\ G_1 \end{bmatrix} \tag{3.43}$$

is square and non-singular. Define $W_2 = G_2$; we now have

$$W = \begin{bmatrix} W_1 \\ W_2 \end{bmatrix}. \tag{3.44}$$

Given this partitioning of W, we can rewrite the normal equations as

$$(W_1^T W_1 + W_2^T W_2)y = W^T \hat{b}. \tag{3.45}$$

Now precondition with W_1 in the standard way:

$$W_1^{-T}(W_1^T W_1 + W_2^T W_2)W_1^{-1}W_1 y = W_1^{-T} W^T \hat{b}, \tag{3.46}$$

or

$$(I + C^T C)w = W_1^{-T} W^T \hat{b}, \quad \text{where } C = W_2 W_1^{-1}, \ \ w = W_1 y. \tag{3.47}$$

We can now apply the conjugate gradient algorithm to this system, leaving the coefficient matrix in factored form. We refer to this method of solving problem LSE as the **preconditioned weighting method, or Pwgt.**

Superficially, the system in (3.47) appears to mimic the BNP system (3.21). Remember, however, that this is not a reduced order problem: the size of the system remains $n \times n$. Still, there is a relationship between BNP and the preconditioned weighting method. We will make the connection explicit in the next chapter.

4. Comparison of the Algorithms

In chapter 3, we introduced algorithm BNP as it appears in [4], and outlined three other iterative algorithms for solving problem LSE: p-cyclic SOR, block AOR, and a preconditioned form of the weighting method. We are now interested in comparing algorithm BNP to each of the other three methods.

The relationship between BNP and p-cyclic SOR is already well understood: despite the elegant convergence theory for p-cyclic SOR, BNP is superior in exact arithmetic. This was established by Freund [11] for unconstrained least squares problems, and by Barlow, Nichols and Plemmons [4] for equality constrained problems. More precisely, if $y^{(0)}$, $r_2^{(0)} = c_2 - G_2 y^{(0)}$, and $r_1^{(0)} = c_1 - G_1 y^{(0)}$ serve as initial iterates for both BNP and 2-cyclic SOR, then the iterates at subsequent steps satisfy the inequality

$$\|c - Gy_{BNP}^{(k)}\|_2 \leq \|c - Gy_{SOR}^{(k+1)}\|_2. \tag{4.1}$$

While the authors state their result explicitly only for 2-cyclic SOR, it holds for 3-cyclic SOR as well. This is a corollary to the main result in §4.2, but it is no surprise: we have already mentioned (see [24]) that the asymptotic convergence of 2-cyclic SOR is better than the 3-cyclic approach.

In this chapter, we consider the relationship of algorithm BNP to the other two types of iterative methods outlined in chapter 3. Adapting the arguments of both Freund and the authors of BNP, we extend their work to show that BNP is also superior to block AOR. We prove this for unconstrained problems in §4.1, and for constrained problems in §4.2. Then, in §4.3 we show that algorithm BNP may be viewed as the limiting case of the preconditioned weighting method.

Note that all the methods require the same pre-iteration processing (factoring E and F, and forming the preconditioner by augmenting the factored equilibrium matrix) and essentially the same computational effort per iteration (two triangular solves and two matrix-vector products dominate the computational effort). Thus, comparing the total number of iterations needed to achieve convergence is a meaningful way to compare the relative performance of these methods.

4.1 BNP vs AOR: Ordinary Least Squares

Our goal is to establish that BNP applied to problem LSE is superior to block AOR in exact arithmetic, by proving a result analogous to equation (4.1). Since optimal 2-AOR coincides with optimal 2-cyclic SOR, it suffices to consider 3-AOR. The proof follows the spirit of the arguments in Freund [11] and Barlow, Nichols, and Plemmons [4], and proceeds in three steps:

1. Given an ordinary least squares problem (no constraints), we consider algorithm BNP applied to the problem, and examine the properties of the iterates.

2. Given the same ordinary least squares problem considered in step 1, we study the properties of 3-AOR applied to the problem, and establish that algorithm BNP is superior to 3-AOR for this unconstrained problem.

3. Given an equality constrained least squares problem, we construct a related ordinary least squares problem. We then establish a relationship between the iterates generated by BNP applied to the constrained problem and the iterates generated by BNP applied to the unconstrained problem. We do the same for 3-AOR as well. Then the results of step 2 allow us to conclude that BNP is superior to 3-AOR for problem LSE.

We complete steps 1 and 2 in this section. In the next section, we relate the constrained problem to an ordinary least squares problem to complete step 3. To prevent confusion, we use a tilde ($\tilde{}$) over many of the quantities associated with the unconstrained problem of this section.

Begin with the ordinary least squares problem

$$\text{minimize} \quad \|\tilde{A}x - \tilde{b}\|_2, \tag{4.2}$$

where $\tilde{A}$ has full column rank. View this problem as a "constrained" least squares problem with zero constraints. The choice of x as the unknown will prove convenient in the next section.

By analogy with (3.2), reorder the rows of $\tilde{A}$ and $\tilde{b}$ as necessary, and partition the coefficient matrix

$$\tilde{A} = \begin{bmatrix} \tilde{A}_1 \\ \tilde{A}_2 \end{bmatrix} \tag{4.3}$$

so that $\tilde{A}_1$ is square and non-singular. Partition the vector

$$\tilde{b} = \begin{bmatrix} \tilde{b}_1 \\ \tilde{b}_2 \end{bmatrix} \tag{4.4}$$

compatibly. The resulting modified Kuhn-Tucker equations are similar to those in (3.4), except the identity matrix replaces the matrix K, and there is no

Lagrange multiplier:

$$\begin{bmatrix} \bar{A}_1 & 0 & I \\ \bar{A}_2 & I & 0 \\ 0 & \bar{A}_2^T & \bar{A}_1^T \end{bmatrix} \begin{bmatrix} x \\ \bar{r}_2 \\ \bar{r}_1 \end{bmatrix} = \begin{bmatrix} \bar{b}_1 \\ \bar{b}_2 \\ 0 \end{bmatrix}. \tag{4.5}$$

Note that BNP and 3-AOR as outlined in §3.2 and §3.3 are perfectly well defined for this system. To obtain the unconstrained equivalent of the BNP system (3.21), simply replace $\bar{K}$ with the identity matrix I, and substitute $\bar{A}_1$, $\bar{A}_2$, $\bar{b}_1$, $\bar{b}_2$, $\bar{r}_1$, $\bar{r}_2$, and x for A_1, A_2, b_1, c_2, r_1, r_2, and y respectively. In particular, by analogy with equation (3.20), note that

$$\bar{Y} = \bar{A}_2\bar{A}_1^{-1}. \tag{4.6}$$

The BNP system for the unconstrained problem can then be written

$$(I + \bar{Y}^T\bar{Y})\bar{r}_1 = \bar{h}, \text{ where } \bar{h} = \bar{Y}^T(\bar{Y}\bar{b}_1 - \bar{b}_2). \tag{4.7}$$

Now apply the conjugate gradient algorithm as given in §3.2. For convenience, we describe the algorithm explicitly in table 4.1.

The AOR algorithm is just as simple to modify, but the absence of x_3 changes its appearance somewhat. See table 4.2 for an explicit description of the algorithm.

We are now prepared to proceed with the proof. In the discussion below, we use the following notation: if w is a vector, and S a set of vectors, then $w + S$ represents the set $\{w + s : s \in S\}$. Similarly, if B is a matrix, then BS is the set $\{Bs : s \in S\}$. If S is convex, so are $w + S$ and BS; if S is a vector subspace, BS is a subspace as well.

Table 4.1: Algorithm BNP (Unconstrained Least Squares)

```
1. Initialize:
    (a) $x^{(0)}$ arbitrary    (normally $x^{(0)} = \tilde{A}_1^{-1}\tilde{b}_1$)
    (b) $\tilde{r}_1^{(0)} = \tilde{b}_1 - \tilde{A}_1 x^{(0)}$    (normally $\tilde{r}_1^{(0)} = 0$)
    (c) $\tilde{r}_2^{(0)} = \tilde{b}_2 - \tilde{A}_2 x^{(0)}$
    (d) $\tilde{\nu}_0 = \tilde{r}_1^{(0)} + \tilde{Y}^T \tilde{r}_2^{(0)}$    ($\tilde{\nu}_k$ is the defect $(I + \tilde{Y}^T\tilde{Y})\tilde{r}_1^{(k)} - \tilde{h}$)
    (e) $\tilde{s}_0 = \tilde{\nu}_0$    ($\tilde{s}_k$ is the direction vector)
2. For $k = 0, 1, \ldots,$ until $\tilde{\nu}_k^T\tilde{\nu}_k <$ tolerance:
    (a) $\tilde{\gamma}_k = \tilde{\nu}_k^T\tilde{\nu}_k / \tilde{s}_k^T(I + \tilde{Y}^T\tilde{Y})\tilde{s}_k$
    (b) $\tilde{r}_1^{(k+1)} = \tilde{r}_1^{(k)} - \tilde{\gamma}_k\tilde{s}_k$
    (c) $x^{(k+1)} = x^{(k)} + \tilde{\gamma}_k\tilde{A}_1^{-1}\tilde{s}_k$
    (d) $\tilde{r}_2^{(k+1)} = \tilde{r}_2^{(k)} - \tilde{\gamma}_k\tilde{Y}\tilde{s}_k$
    (e) $\tilde{\nu}_{k+1} = \tilde{r}_1^{(k+1)} + \tilde{Y}^T\tilde{r}_2^{(k+1)}$
    (f) $\tilde{\beta}_{k+1} = \tilde{\nu}_{k+1}^T\tilde{\nu}_{k+1} / \tilde{\nu}_k^T\tilde{\nu}_k$
    (g) $\tilde{s}_{k+1} = \tilde{\nu}_{k+1} + \tilde{\beta}_{k+1}\tilde{s}_k$
```

Table 4.2: Three-Block AOR (Unconstrained Least Squares)

```
1. Initialize:
    (a) $x^{(0)}$ arbitrary
    (b) $\bar{r}_2^{(0)} = \bar{b}_2 - \bar{A}_2 x^{(0)}$
    (c) $\bar{r}_1^{(0)} = \bar{b}_1 - \bar{A}_1 x^{(0)}$
2. For $k = 0, 1, \ldots$, until convergence:
    (a) $x^{(k+\frac{1}{2})} = \bar{A}_1^{-1}(\bar{b}_1 - \bar{r}_1^{(k)})$
    (b) $\bar{r}_2^{(k+\frac{1}{2})} = \bar{b}_2 - \bar{A}_2 \left[(1-\beta)x^{(k)} + \beta x^{(k+\frac{1}{2})}\right]$
    (c) $\bar{r}_1^{(k+\frac{1}{2})} = -\bar{A}_1^{-T} \bar{A}_2^T \left[(1-\beta)\bar{r}_2^{(k)} + \beta \bar{r}_2^{(k+\frac{1}{2})}\right]$
    (d) $x^{(k+1)} = (1-\omega)x^{(k)} + \omega x^{(k+\frac{1}{2})}$
    (e) $\bar{r}_2^{(k+1)} = (1-\omega)\bar{r}_2^{(k)} + \omega \bar{r}_2^{(k+\frac{1}{2})}$
    (f) $\bar{r}_1^{(k+1)} = (1-\omega)\bar{r}_1^{(k)} + \omega \bar{r}_1^{(k+\frac{1}{2})}$
```

Let $x^{(0)}$, $\tilde{r}_1^{(0)} = \tilde{b}_1 - \tilde{A}_1 x^{(0)}$, and $\tilde{r}_2^{(0)} = \tilde{b}_2 - \tilde{A}_2 x^{(0)}$ define a single set of initial iterates for both BNP and 3-AOR. Define the following vectors in $\mathcal{R}^n$:

$$\begin{aligned} \hat{v}_0 &= \tilde{b}_1 + \tilde{Y}^T \tilde{b}_2 && (4.8) \\ v_0 &= \hat{v}_0 - (\tilde{A}_1 + \tilde{Y}^T \tilde{A}_2) x^{(0)} && (4.9) \end{aligned}$$

$$\begin{aligned} \hat{w}_0 &= \tilde{A}_1^{-1} \hat{v}_0 && (4.10) \\ &= \tilde{A}_1^{-1} (\tilde{b}_1 + \tilde{Y}^T \tilde{b}_2) \\ w_0 &= \tilde{A}_1^{-1} v_0 && (4.11) \\ &= \hat{w}_0 - \tilde{A}_1^{-1} (\tilde{A}_1 + \tilde{Y}^T \tilde{A}_2) x^{(0)}. \end{aligned}$$

Additionally, let

$$\tilde{B} = \tilde{A}_1^{-1} \tilde{Y}^T \tilde{Y} \tilde{A}_1 = \tilde{A}_1^{-1} \tilde{Y}^T \tilde{A}_2, \qquad (4.12)$$

noting that

$$w_0 = \hat{w}_0 - (I + \tilde{B}) x^{(0)}. \qquad (4.13)$$

Finally, define a sequence of Krylov subspaces:

$$\begin{aligned} S_0 &= \{0\} \\ S_k &= \operatorname{span}\left\{ w_0, \tilde{B} w_0, \ldots, \tilde{B}^{k-1} w_0 \right\}. && (4.14) \end{aligned}$$

Lemma 4.1 (Freund) *Let $x^{(0)}$, $\tilde{r}_2^{(0)} = \tilde{b}_2 - \tilde{A}_2 x^{(0)}$, and $\tilde{r}_1^{(0)} = \tilde{b}_1 - \tilde{A}_1 x^{(0)}$ be initial iterates for algorithm BNP applied to the ordinary least squares problem (4.2). Then the kth iterate $x^{(k)}$ lies in the set $x^{(0)} + S_k$. Moreover, $x^{(k)}$ minimizes the residual $\|\tilde{r}\|_2 = \|\tilde{b} - \tilde{A} x\|_2$ over all x in $x^{(0)} + S_k$.*

Proof. See [11]. □

Theorem 4.1 *Let $x^{(0)}$, $\tilde{r}_2^{(0)} = \tilde{b}_2 - \tilde{A}_2 x^{(0)}$, and $\tilde{r}_1^{(0)} = \tilde{b}_1 - \tilde{A}_1 x^{(0)}$ be initial iterates for both BNP and 3-AOR applied to the ordinary least squares problem (4.2). Let $x_{BNP}^{(k)}$ and $x_{AOR}^{(k)}$ represent the kth iterates generated by the two algorithms respectively, and define $\tilde{r}_{BNP}^{(k)} = \tilde{b} - \tilde{A}x_{BNP}^{(k)}$ and $\tilde{r}_{AOR}^{(k)} = \tilde{b} - \tilde{A}x_{AOR}^{(k)}$ to be the associated residuals. Then the iterates satisfy the inequality*

$$\|\tilde{r}_{BNP}^{(k)}\|_2 \le \|\tilde{r}_{AOR}^{(k+1)}\|_2$$

in exact arithmetic, regardless of the AOR iteration parameters.

Proof. By Lemma 4.1, it suffices to prove that the AOR iterate $x_{AOR}^{(k+1)}$ lies in $x^{(0)} + S_k$. Suppressing the "AOR" subscripts for simplicity, we accomplish this by establishing the following relationships:

(a) $x^{(k)} \in x^{(0)} + S_{k-1}$

(b) $\tilde{b}_2 - \tilde{r}_2^{(k-\frac{1}{2})} \in \tilde{A}_2(x^{(0)} + S_{k-1})$

(c) $\tilde{b}_2 - \tilde{r}_2^{(k)} \in \tilde{A}_2(x^{(0)} + S_{k-1})$

(d) $\tilde{b}_1 + \tilde{Y}^T \tilde{r}_2^{(k-\frac{1}{2})} \in \hat{v}_0 - \tilde{Y}^T \tilde{A}_2(x^{(0)} + S_{k-1})$

(e) $\tilde{b}_1 + \tilde{Y}^T \tilde{r}_2^{(k)} \in \hat{v}_0 - \tilde{Y}^T \tilde{A}_2(x^{(0)} + S_{k-1})$

(f) $\tilde{b}_1 - \tilde{r}_1^{(k-\frac{1}{2})} \in \hat{v}_0 - \tilde{Y}^T \tilde{A}_2(x^{(0)} + S_{k-1})$

(g) $\tilde{b}_1 - \tilde{r}_1^{(k)} \in \tilde{A}_1 x^{(0)} + \text{span}\{v_0\} + \tilde{A}_2 \tilde{Y}^T S_{k-1}$

(h) $x^{(k+\frac{1}{2})} \in x^{(0)} + S_k$

Our goal is to establish the first of these relationships; the others are means toward that end. We argue by induction. Listing the early iterates explicitly, we have:

$$\begin{aligned} x^{(\frac{1}{2})} &= x^{(0)} \\ \tilde{r}_2^{(\frac{1}{2})} &= \tilde{r}_2^{(0)} \\ \tilde{r}_1^{(\frac{1}{2})} &= -\tilde{Y}^T \tilde{r}_2^{(0)} \end{aligned}$$

$$\begin{aligned} x^{(1)} &= x^{(0)} \\ \tilde{r}_2^{(1)} &= \tilde{r}_2^{(0)} \\ \tilde{r}_1^{(1)} &= \tilde{r}_1^{(0)} - \omega v_0 \end{aligned}$$

$$x^{(1+\frac{1}{2})} = x^{(0)} + \omega w_0.$$

Use these values to confirm that each of the inductive hypotheses holds for $k = 1$. Now assume all eight hypotheses hold for a fixed k, and consider $k + 1$.

Proof of (a): Since $x^{(0)} + S_{k-1}$ is contained in $x^{(0)} + S_k$, inductive assumption (a) tells us that $x^{(k)}$ is an element of $x^{(0)} + S_k$. By assumption (h), $x^{(k+\frac{1}{2})}$ is also in this set. Thus, by convexity, $x^{(k+1)} = (1-\omega)x^{(k)} + \omega x^{(k+\frac{1}{2})}$ is in $x^{(0)} + S_k$ as required.

Proof of (b): Equation 2(b) of table 4.2 tells us that

$$\tilde{b}_2 - \tilde{r}_2^{(k+\frac{1}{2})} = \tilde{A}_2 \left[(1-\beta)x^{(k)} + \beta x^{(k+\frac{1}{2})}\right].$$

By assumption (a), we know $\tilde{A}_2 x^{(k)}$ is contained in $\tilde{A}_2(x^{(0)} + S_{k-1})$, which in turn is a subset of $\tilde{A}_2(x^{(0)} + S_k)$. Moreover, $\tilde{A}_2 x^{(k+\frac{1}{2})}$ is in the latter set by inductive assumption (h). Thus, by convexity, $\tilde{b}_2 - \tilde{r}_2^{(k+\frac{1}{2})}$ is in $\tilde{A}_2(x^{(0)} + S_k)$ as required.

Proof of (c): From assumption (c) and the containment argument used above, we have that $\bar{b}_2 - \bar{r}_2^{(k)}$ is in $\tilde{A}_2(x^{(0)} + S_k)$. The proof of (b) tells us that $\bar{b}_2 - \bar{r}_2^{(k+\frac{1}{2})}$ is in the latter set as well. Rearrange equation 2(e) of table 4.2 to see that $\bar{b}_2 - \bar{r}_2^{(k+1)}$ is a convex combination of $\bar{b}_2 - \bar{r}_2^{(k)}$ and $\bar{b}_2 - \bar{r}_2^{(k+\frac{1}{2})}$, and so is in $\tilde{A}_2(x^{(0)} + S_k)$ as required.

Proof of (d): From the definition of $\hat{v}_0$ in equation (4.8), and the definition of $\bar{r}_2^{(k+\frac{1}{2})}$ in table 4.2, we find that

$$\bar{b}_1 + \tilde{Y}^T \bar{r}_2^{(k+\frac{1}{2})} = \hat{v}_0 - \tilde{Y}^T \tilde{A}_2 \left[(1-\beta)x^{(k)} + \beta x^{(k+\frac{1}{2})}\right].$$

Assumptions (a) and (h) combined with convexity then give us the required result.

Proof of (e) and (f): Similar to the proof of (c).

Proof of (g): By assumption (g), there is a scalar α, and a vector z_1 in $\tilde{Y}^T \tilde{A}_2 S_k$, such that

$$\bar{b}_1 - \bar{r}_1^{(k)} = \tilde{A}_1 x^{(0)} + \alpha v_0 + z_1.$$

By the proof of (f), there is a $z_2 \in \tilde{Y}^T \tilde{A}_2 S_k$ such that

$$\bar{b}_1 - \bar{r}_1^{(k+\frac{1}{2})} = \hat{v}_0 - \tilde{Y}^T \tilde{A}_2 x^{(0)} + z_2.$$

Let $z = (1-\omega)z_1 + \omega z_2$, and note that $z \in \tilde{Y}^T \tilde{A}_2 S_k$. Rearrange equation 2(f) of table 4.2 to obtain

$$\bar{b}_1 - \bar{r}_1^{(k+1)} = (1-\omega)(\bar{b}_1 - \bar{r}_1^{(k)}) + \omega(\bar{b}_1 - \bar{r}_1^{(k+\frac{1}{2})}).$$

Remembering that $v_0 = \hat{v}_0 - (\tilde{A}_1 + \tilde{Y}^T \tilde{A}_2)x^{(0)}$, use the expressions for $\bar{b}_1 - \bar{r}_1^{(k)}$ and $\bar{b}_1 - \bar{r}_1^{(k+\frac{1}{2})}$ to find that

$$\bar{b}_1 - \bar{r}_1^{(k+1)} = \tilde{A}_1 x^{(0)} + \alpha(1-\omega)v_0 + \omega v_0 + z,$$

which is in $\tilde{A}_1 x^{(0)} + \text{span}\{v_0\} + \tilde{Y}^T \tilde{A}_2 S_k$ as required.

Proof of (h): From table 4.2 we have $x^{(k+1+\frac{1}{2})} = \tilde{A}_1^{-1}(\tilde{b}_1 - \tilde{r}_1^{(k+1)})$. By the proof of (g), there exists a scalar γ such that

$$\tilde{b}_1 - \tilde{r}_1^{(k+1)} \in \tilde{A}_1 x^{(0)} + \gamma v_0 + \tilde{Y}^T \tilde{A}_2 S_k.$$

Recalling that $w_0 = \tilde{A}_1^{-1} v_0$, these two facts tell us that $x^{(k+1+\frac{1}{2})}$ is in the set $x^{(0)} + \gamma w_0 + \tilde{A}_1^{-1}\tilde{Y}^T \tilde{A}_2 S_k$. But w_0 is an element of S_{k+1}, and $\tilde{A}_1^{-1}\tilde{Y}^T \tilde{A}_2 S_k = \tilde{B} S_k$, which is a vector subspace of S_{k+1}. Hence $x^{(k+1+\frac{1}{2})}$ is in $x^{(0)} + S_{k+1}$ as required. □

4.2 BNP vs AOR: Constrained Least Squares

In the previous section, we proved that in exact arithmetic, BNP applied to an unconstrained least squares problem converges at least as fast as 3-AOR. To extend this result to constrained least squares problems, we establish a connection between problem LSE and a specially constructed unconstrained problem. The argument below is an adaptation of Theorem 2.1 in Barlow, Nichols, and Plemmons [4], and is based on a special case of the classical nullspace method (see §5.1).

Consider problem LSE as given in equation (2.1), and the associated modified Kuhn-Tucker equations (3.4). Define the matrix

$$N = A_1^{-1} \bar{K} = A_1^{-1} \begin{bmatrix} 0 \\ I \end{bmatrix}. \tag{4.15}$$

Observe that

$$\begin{bmatrix} E \\ G_1 \end{bmatrix} A_1^{-1} = A_1 A_1^{-1} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}. \tag{4.16}$$

This means $EA_1^{-1} = \begin{bmatrix} I & 0 \end{bmatrix}$, so $EN = 0$. Moreover, N has $n-m_1$ linearly independent columns, so the columns of N form a basis for the nullspace of E.

By a similar argument, the vector

$$y_p = A_1^{-1}b_1 \tag{4.17}$$

is a particular solution of the constraint $Ey = b$. Note that any vector satisfying the constraint can be written as $y = y_p + Nx$ for some choice of x, and minimizing $\|Gy - c\|_2$ subject to the constraint amounts to minimizing $\|G(y_p + Nx) - c\|_2$ over all possible x. The latter minimization is an unconstrained problem of order $n-m_1$. Isolating the unknown x, we have the ordinary least squares problem

$$\text{minimize} \quad \|\tilde{A}x - \tilde{b}\|_2 \quad \text{where} \quad \tilde{A} = GN, \ \tilde{b} = c - Gy_p. \tag{4.18}$$

Referring to equation (4.16), we find that

$$GN = \begin{bmatrix} G_1N \\ G_2N \end{bmatrix} = \begin{bmatrix} I \\ Y \end{bmatrix}, \tag{4.19}$$

where $Y = A_2A_1^{-1}\tilde{K}$ is as in the BNP system (3.21). Additionally,

$$G_1y_p = G_1A_1^{-1}b_1 = \tilde{K}^T \begin{bmatrix} b \\ c_1 \end{bmatrix} = c_1, \tag{4.20}$$

so $c_1 - G_1y_p = 0$. All of these relationships give us an elegant form for the modified Kuhn-Tucker system associated with the ordinary least squares problem given in (4.18):

$$\begin{bmatrix} I & 0 & I \\ Y & 0 & 0 \\ 0 & Y^T & I \end{bmatrix} \begin{bmatrix} x \\ \tilde{r}_2 \\ \tilde{r}_1 \end{bmatrix} = \begin{bmatrix} 0 \\ c_2 - G_2y_p \\ 0 \end{bmatrix}. \tag{4.21}$$

In terms of the notation from the previous section, we have

$$\tilde{A}_1 = I \tag{4.22}$$

$$\tilde{A}_2 = Y \tag{4.23}$$

$$\tilde{b}_1 = 0 \tag{4.24}$$

$$\tilde{b}_2 = c_2 - A_2 A_1^{-1} b_1. \tag{4.25}$$

Now let $y^{(0)}$ be an arbitrary initial iterate for both BNP and 3-AOR applied to problem LSE. There is a unique $x^{(0)}$ such that $y^{(0)} = y_p + Nx^{(0)}$; in fact, the correct value is given by

$$x^{(0)} = \bar{K}^T (A_1 y^{(0)} - b_1). \tag{4.26}$$

Use this value of $x^{(0)}$ as the initial iterate for BNP and 3-AOR applied to the ordinary least squares problem (4.18). We now define the corresponding initial residual iterates, and relate the subsequent iterates for the constrained problem to those of the unconstrained problem.

Lemma 4.2 *Let $y^{(0)}$, $r_2^{(0)} = c_2 - A_2 y^{(0)}$, and $r_1^{(0)} = c_1 - G_1 y^{(0)}$ be initial iterates for BNP applied to problem LSE. Define $x^{(0)}$ as in equation (4.26), and let $x^{(0)}$, $\tilde{r}_2^{(0)} = \tilde{b}_2 - \tilde{A}_2 x^{(0)}$, and $\tilde{r}_1^{(0)} = \tilde{b}_1 - \tilde{A}_1 x^{(0)}$ be initial iterates for BNP applied to the related ordinary least squares problem defined in equations (4.18) through (4.25). Then subsequent iterates satisfy $y^{(k)} = y_p + Nx^{(k)}$.*

Proof. Refer to tables 3.1 and 4.1. Remembering that a tilde ($\tilde{\ }$) represents a quantity associated with the ordinary least squares problem, we prove by induction that $y^{(k)} = y_p + Nx^{(k)}$, $r_2^{(k)} = \tilde{r}_2^{(k)}$, $r_1^{(k)} = \tilde{r}_1^{(k)}$, $\nu_k = \tilde{\nu}_k$, and $s_k = \tilde{s}_k$ for all k.

First consider $k = 0$. We know $y^{(0)} = y_p + Nx^{(0)}$ by the definition of $x^{(0)}$. Since $\bar{A}_1 = I$ and $\bar{b}_1 = 0$, we have $\bar{r}_1 = -x^{(0)}$, which is $-\bar{K}^T(A_1y^{(0)} - b_1)$. But $\bar{K}^T A_1 = G_1$ and $\bar{K}^T b_1 = c_1$, so $\bar{r}_1^{(0)} = c_1 - G_1 y^{(0)}$, which is $r_1^{(0)}$ as required. Additionally, simple substitution tells us that $\bar{r}_2^{(0)}$ is $c_2 - A_2(y_p + Nx^{(0)})$. This is $c_2 - A_2 y^{(0)}$, which is $r_2^{(0)}$ as required. Finally, note that the initial defects and initial direction vectors satisfy $\nu_0 = \bar{\nu}_0$ and $s_0 = \bar{s}_0$ trivially.

Now assume $y^{(k)} = y_p + Nx^{(k)}$, $r_2^{(k)} = \bar{r}_2^{(k)}$, $r_1^{(k)} = \bar{r}_1^{(k)}$, $\nu_k = \bar{\nu}_k$, and $s_k = \bar{s}_k$ are all true for a fixed k, and consider k+1. When BNP is applied to the ordinary least squares problem, we have $\bar{Y} = \bar{A}_2\bar{A}_1^{-1} = Y$. This, combined with the inductive assumptions, gives us $\gamma_k = \bar{\gamma}_k$ immediately, from which we get $r_2^{(k+1)} = \bar{r}_2^{(k+1)}$, $r_1^{(k+1)} = \bar{r}_1^{(k+1)}$, and $\nu_{k+1} = \bar{\nu}_{k+1}$. We can then conclude that $\beta_{k+1} = \bar{\beta}_{k+1}$, from which we obtain $s_{k+1} = \bar{s}_{k+1}$ as required.

Finally, we know $y^{(k+1)} = y^{(k)} + \gamma_k A_1^{-1}\bar{K}s_k$. Recalling that $\bar{A}_1 = I$, we have $A_1^{-1}\bar{K} = \bar{A}_1^{-1}N$. Moreover, $s_k = \bar{s}_k$ and $y^{(k)} = y_p + Nx^{(k)}$ by hypothesis. So $y^{(k+1)} = y_p + N(x^{(k)} + \gamma_k\bar{A}_1^{-1}\bar{s}_k)$, which is $y_p + Nx^{(k+1)}$ as required. □

Lemma 4.3 *Let $y^{(0)}$, $r_2^{(0)} = c_2 - A_2y^{(0)}$, $r_1^{(0)} = c_1 - G_1y^{(0)}$, and $\lambda^{(0)}$ be initial iterates for 3-AOR applied to problem LSE. Define $x^{(0)}$ as in equation (4.26), and let $x^{(0)}$, $\bar{r}_2^{(0)} = \bar{b}_2 - \bar{A}_2x^{(0)}$, and $\bar{r}_1^{(0)} = \bar{b}_1 - \bar{A}_1x^{(0)}$ be initial iterates for 3-AOR (with the same choice of ω and β) applied to the related ordinary least squares problem defined by equations (4.18) through (4.25). Then subsequent iterates satisfy $y^{(k)} = y_p + Nx^{(k)}$.*

Proof. Refer to the description of the algorithms given in tables 3.2 and 4.2. We establish by induction that $y^{(k)} = y_p + Nx^{(k)}$, $r_2^{(k)} = \bar{r}_2^{(k)}$, and $r_1^{(k)} = \bar{r}_1^{(k)}$. The case $k = 0$ is in the proof of Lemma 4.2.

Now assume $y^{(k)} = y_p + Nx^{(k)}$, $r_2^{(k)} = \tilde{r}_2^{(k)}$, and $r_1^{(k)} = \tilde{r}_1^{(k)}$ are true for a fixed k, and consider $k+1$. Since $\tilde{A}_1 = I$ and $\tilde{b} = 0$, we know that $x^{(k+\frac{1}{2})} = -\tilde{r}_1^{(k)}$ for all k. By the inductive assumptions, this means $x^{(k+\frac{1}{2})} = -r_1^{(k)}$. After this substitution, the defining equation $y^{(k+\frac{1}{2})} = A_1^{-1}(b_1 - \bar{K}r_1^{(k)})$ becomes $y^{(k+\frac{1}{2})} = y_p + Nx^{(k+\frac{1}{2})}$. We also know that $y^{(k)} = y_p + Nx^{(k)}$ by inductive assumption. Thus, we find that the equation $y^{(k+1)} = (1-\omega)y^{(k)} + \omega y^{(k+\frac{1}{2})}$ becomes

$$y^{(k+1)} = y_p + N\left\{(1-\omega)x^{(k)} + \omega x^{(k+\frac{1}{2})}\right\},$$

which is $y_p + Nx^{(k+1)}$ as required.

We obtain $r_2^{(k+1)} = \tilde{r}_2^{(k+1)}$ by a straightforward substitution. Observing that $r_1^{(k)} = \bar{K}^T z_3^{(k)}$, a simple substitution produces $r_1^{(k+1)} = \tilde{r}_1^{(k+1)}$ as well. □

Lemma 4.4 *Let $y^{(0)}$, $r_2^{(0)} = c_2 - A_2 y^{(0)}$, and $r_1^{(0)} = c_1 - G_1 y^{(0)}$ be initial iterates for either BNP or 3-AOR applied to problem LSE (the choice of $\lambda^{(0)}$ in 3-AOR is immaterial). Define $x^{(0)}$ as in equation (4.26), and let $x^{(0)}$, $\tilde{r}_2^{(0)} = \tilde{b}_2 - \tilde{A}_2 x^{(0)}$, and $\tilde{r}_1^{(0)} = \tilde{b}_1 - \tilde{A}_1 x^{(0)}$ be initial iterates for the same algorithm applied to the related ordinary least squares problem defined by equations (4.18) through (4.25). Then the residuals $c - Gy^{(k)}$ and $\tilde{b} - \tilde{A}x^{(k)}$ are equal.*

Proof. By Lemmas 4.2 and 4.3, we know that $y^{(k)} = y_p + Nx^{(k)}$. This means that $c - Gy^{(k)} = (c - Gy_p) - GNx^{(k)}$. But $GN = \tilde{A}$ and $c - Gy_p = \tilde{b}$ by (4.18), so the result follows. □

Theorem 4.2 *Let $y^{(0)}$, $r_2^{(0)} = c_2 - A_2 y^{(0)}$, and $r_1^{(0)} = c_1 - G_1 y^{(0)}$ be initial iterates for BNP applied to problem LSE. Let the same quantities with an arbitrary $\lambda^{(0)}$ serve as initial iterates for 3-AOR. Let $y_{BNP}^{(k)}$ and $y_{AOR}^{(k)}$ represent* *the kth solution iterates generated by the two algorithms respectively, and define $r_{BNP}^{(k)} = c - Gy_{BNP}^{(k)}$ and $r_{AOR}^{(k)} = c - Gy_{AOR}^{(k)}$ to be the associated residuals. Then the iterates satisfy the inequality*

$$\|r_{BNP}^{(k)}\|_2 \leq \|r_{AOR}^{(k+1)}\|_2$$

in exact arithmetic, regardless of the AOR iteration parameters.

Proof. Apply BNP and 3-AOR to the related ordinary least squares problem defined by equations (4.18) through (4.25). By Lemma 4.4 we have

$$r_{BNP}^{(k)} = \tilde{b} - \tilde{A}x_{BNP}^{(k)}$$

$$r_{AOR}^{(k+1)} = \tilde{b} - \tilde{A}x_{AOR}^{(k+1)}.$$

The result then follows from Theorem 4.1. □

While the theorem applies to calculations performed in exact arithmetic, it suggests that BNP should outperform 3-AOR on problem LSE. Our numerical experiments (see chapter 6) suggest that this is in fact the case.

Finally, we obtain as a corollary the fact that algorithm BNP is superior to optimal 3-cyclic SOR. Since we know that optimal 2-cyclic SOR is asymptotically faster than 3-cyclic SOR for problem LSE, this merely formalizes what one would expect.

Corollary 4.1 *Let $y^{(0)}$, $r_2^{(0)} = c_2 - A_2y^{(0)}$, and $r_1^{(0)} = c_1 - G_1y^{(0)}$ be initial iterates for BNP applied to problem LSE. Let the same quantities with an arbitrary $\lambda^{(0)}$ serve as initial iterates for 3-cyclic SOR. Let $y_{BNP}^{(k)}$ and $y_{SOR}^{(k)}$ represent the kth solution iterates generated by the two algorithms respectively, and define*

$r^{(k)}_{BNP} = c - Gy^{(k)}_{BNP}$ and $r^{(k)}_{SOR} = c - Gy^{(k)}_{SOR}$ to be the associated residuals. Then the iterates satisfy the inequality

$$\|r^{(k)}_{BNP}\|_2 \le \|r^{(k+1)}_{SOR}\|_2$$

in exact arithmetic, regardless of the value of ω.

Proof. The result in theorem 4.2 hold regardless of the AOR parameters. In particular, it holds for the special case $\omega = \beta$, which is 3-cyclic SOR. □

4.3 BNP and Preconditioned Weighting

It is at least mildly surprising that algorithm BNP is also closely related to the preconditioned weighting method described in §3.3; we detail the connection in this section. The analysis holds when the preconditioner W_1 is formed by interlacing, but the permutation matrices used to define the interlacing unnecessarily complicate the notation. For that reason, we proceed using the augmentation technique based on the trapezoidal matrix E_t (§3.1).

Following the notation in §3.3, we begin with the matrix $W = \begin{bmatrix} \tau E \\ G \end{bmatrix}$ and the vector $\hat{b} = \begin{bmatrix} \tau b \\ c \end{bmatrix}$. We then apply the conjugate gradient algorithm to the factored linear system

$$(I + C^T C)w = W_1^{-T} W^T \hat{b}, \tag{4.27}$$

where $W_1 = \begin{bmatrix} \tau E \\ G_1 \end{bmatrix}$ and $W_2 = G_2$ are the upper and lower blocks of W respectively, C is the matrix $W_2 W_1^{-1}$, and the vector $w = W_1 y$ is the new unknown. Recall from §3.3 that W_1 is non-singular.

Now let E be the trapezoidal matrix $E_t = \begin{bmatrix} E_L & E_R \end{bmatrix}$ defined in equation (3.6). If we form W_1 and W_2 using the augmentation technique described in §3.1, we obtain

$$W_1 = \begin{bmatrix} \tau E_L & \tau E_R \\ & G_{12} \end{bmatrix}, \tag{4.28}$$

$$W_2 = \begin{bmatrix} G_{21} & G_{22} \end{bmatrix}, \tag{4.29}$$

where G is given by

$$G = \begin{bmatrix} G_{21} & G_{22} \\ & G_{12} \end{bmatrix}. \tag{4.30}$$

In this case, the righthand side $W_1^{-T} W^T \hat{b}$ becomes

$$W_1^{-T} W^T \hat{b} = \begin{bmatrix} \tau b \\ c_1 \end{bmatrix} + C^T c_2. \tag{4.31}$$

We can now explicitly calculate the inverse of W_1:

$$W_1^{-1} = \begin{bmatrix} \frac{1}{\tau} E_L^{-1} & -E_L^{-1} E_R G_{12}^{-1} \\ & G_{12}^{-1} \end{bmatrix}. \tag{4.32}$$

This in turn allows us to rewrite the preconditioned system (4.27) as

$$\begin{bmatrix} (I + \tau^{-2} V^T V) & \tau^{-1} V^T Y \\ \tau^{-1} Y^T V & (I + Y^T Y) \end{bmatrix} \begin{bmatrix} w_L \\ w_R \end{bmatrix} = \begin{bmatrix} \hat{h}_1 \\ \hat{h}_2 \end{bmatrix} \tag{4.33}$$

where

$$w = \begin{bmatrix} w_L \\ w_R \end{bmatrix}, \tag{4.34}$$

$$V = G_{21} E_L^{-1}, \tag{4.35}$$

$$\begin{bmatrix} \hat{h}_1 \\ \hat{h}_2 \end{bmatrix} = \begin{bmatrix} \tau b + \frac{1}{\tau} V^T c_2 \\ c_1 + Y^T c_2 \end{bmatrix}, \tag{4.36}$$

and Y is as in equation (3.20). In the limit as $\tau \rightarrow \infty$, the coefficient matrix is

$$\begin{bmatrix} I & \\ & (I + Y^T Y) \end{bmatrix}. \tag{4.37}$$

Loosely speaking, therefore, one would expect the conjugate gradient method with the coefficient matrix given in (4.33) to perform as if the matrix was "almost" block diagonal. The algorithm should produce w_L in a very small number of iterations (the associated diagonal block is nearly the identity matrix), and it should act like algorithm BNP on the lower block of the system. In a sense, then, BNP may be viewed as the limiting case of the preconditioned weighting method. This is true despite the fact that PWgt solves an $n \times n$ system rather than a reduced order problem. In the limit as $\tau \to \infty$, the m_1 leading unknowns in the PWgt system are determined by what amounts to a trivial linear system, and the method acts as if it were an order-reducing algorithm applied to the lower $(n - m_1)$ unknowns.

When formalizing the connection between the two schemes, however, there is one subtle point to note. It is tempting to start with the block system in (4.33), delete terms involving negative powers of τ, and expect to recover algorithm BNP. It would quickly become clear that this does not work.

The problem is this: while the lower left block of the coefficient matrix is $O(\tau^{-1})$, the vector w_L is $O(\tau)$. Thus the product of these two components is $O(1)$ and cannot be ignored. To see why this is so, recall that $w = W_1 y$ by definition, which means

$$\begin{bmatrix} w_L \\ w_R \end{bmatrix} = \begin{bmatrix} \tau E_t \\ G_1 \end{bmatrix} y. \tag{4.38}$$

Thus

$$w_L = \tau E_t y = \tau b, \tag{4.39}$$

and there is no need to compute w_L at all.

But there is also a natural interpretation of the lower block w_R. From (4.38) as well as the definition of r_1 given in equations (2.6) and (3.4), we have

$$w_R = G_1 y = c_1 - r_1. \tag{4.40}$$

Now, armed with a clearer picture of what the blocks in w represent, we can use (4.39) and (4.40) to produce algorithm BNP from the weighting method.

Begin with the second block equation in (4.33):

$$\frac{1}{\tau} Y^T G_{21} E_L^{-1} w_L + (I + Y^T Y) w_R = c_1 + Y^T c_2. \tag{4.41}$$

Now substitute (4.39) and (4.40) into this equation, and rearrange to obtain

$$(I + Y^T Y) r_1 = Y^T (Y c_1 + G_{21} E_L^{-1} b - c_2). \tag{4.42}$$

Note that the substitutions have eliminated τ entirely; in fact, minor simplification of the righthand side produces the BNP system (3.21).

Of course, the BNP unknown and righthand side are different than those for the preconditioned weighting method, so we can expect some variation in relative performance from one problem to another. Moreover, each method has its own advantages and disadvantages. BNP, for example, is a "parameter-free" algorithm, while PWgt requires the user to specify τ: if τ is too small, the solution to the weighted problem will be a poor approximation to the solution of problem LSE; if τ is too large, then underflow, overflow, or roundoff errors will degrade accuracy or prevent normal termination. A single PWgt iteration, on the other hand, is generally slightly faster than a corresponding BNP iteration: BNP requires gather/scatter vector operations (multiplications with permutations of $\bar{K}$ and $\bar{K}^T$) that are not needed in the weighting algorithm. We demonstrate in chapter 6 that the algorithms do indeed perform comparably, but that neither is clearly superior to the other.

5. Implicit Nullspace Methods

Algorithm BNP as originally proposed has at least two limitations we wish to overcome. The first concerns the conditions under which we can successfully construct a suitable preconditioning matrix A_1. Theoretically, assumptions H1 and H2 in §2.1 are sufficient: if E has full row rank, and $\begin{bmatrix} E \\ G \end{bmatrix}$ have full column rank, then one can find rows of G to construct G_1 so that $A_1 = \begin{bmatrix} E \\ G_1 \end{bmatrix}$ is non-singular. In practice, though, it may be quite difficult to determine the correct rows of G to use [5]. Moreover, even if we can identify suitable rows of G, the resulting A_1 may not be easily invertible.

As long as G itself has full column rank and relatively simple structure, we can successfully produce an upper triangular A_1 by interlacing (§3.1). Suppose, though, that G lacks full column rank. Then, given a specified column of G, there may be no row of G with leading non-zero in that column, even after orthogonal reduction. If we need such a row to augment the stairstep matrix E_s, we will not be able to produce an upper triangular A_1 by interlacing.

Even if G has full column rank, we may encounter difficulties forming A_1 by interlacing. Suppose it is impractical to reduce G to upper triangular form (this is true, for example, in the Stokes problem described in §2.3). In this case, interlacing can only succeed if the unreduced (or partially reduced) matrix G has rows with leading non-zeros in all columns in which E_s does not have leading non-zeros. Fortunately, G as given in §2.3 has rows with leading non-zeros in every possible column, but this is merely a consequence of the simple geometry we chose for the domain.

The second difficulty concerns problems expressed in the saddle point form given in (2.9); again, the Stokes model is a good example. Here the vector c, which is needed to form the righthand side of the BNP system (3.21), is not generally available; instead, the vector $s = -G^T c$ is specified. Depending on the structure of G (if G is even known), it may or may not be practical to determine c explicitly.

One way around this obstacle is to attempt to rewrite the righthand side of the BNP system in terms of s rather than c. But this is not possible for the problem as formulated in (3.21), because the righthand side lacks a critical term involving c_1 (the term is buried in the unknown r_1). Another approach, of course, is to use a different algorithm. The preconditioned weighting algorithm is one possibility, and a little algebra does allow us to rewrite the righthand side of the PWgt system (3.47) in terms of s rather than c:

$$W_1^{-T} W^T \begin{bmatrix} \tau b \\ c \end{bmatrix} = \begin{bmatrix} \tau b \\ 0 \end{bmatrix} - W_1^{-T} s. \tag{5.1}$$

But even this doesn't overcome the difficulty completely. We still need to construct W_1 as defined in (3.43), and this requires having G in a form suitable for use in the interlacing scheme. We would like to produce an order-reducing conjugate gradient algorithm which can solve saddle point problems in which neither c nor G is readily available.

In this chapter we extend algorithm BNP to a class of methods capable of dealing with each of these difficulties. Our approach is based on the classical

nullspace method, which makes use of a matrix N (the **basis matrix**) whose columns form a basis for the nullspace of the equilibrium matrix. The technique used in §4.3 to relate problem LSE to an unconstrained problem is a special case of the nullspace method.

In §5.1 we describe the nullspace method, and show that algorithm BNP may be viewed as a variation of this method for a certain distinguished choice of the basis matrix. In BNP, however, the basis matrix is used but never explicitly formed; thus algorithm BNP becomes an example of a class of methods we call **implicit nullspace methods**. We describe in §5.2 a technique for producing implicit nullspace methods for other choices of the basis matrix, and outline a general algorithm suitable for implementation. Then, in §5.3, we propose specific examples of implicit nullspace methods, emphasizing ways to overcome the difficulties described above. In the next chapter, we will report on experiments with each of these methods, using test problems based on both the static analysis of engineering structures and Stokes flow.

5.1 BNP as an Implicit Nullspace Method

The original derivation of algorithm BNP, reported in [4] and detailed in §3.1, involves applying block elimination to the modified Kuhn-Tucker equations. In this section, we derive the method in a new way, establishing a connection between algorithm BNP and the classical **nullspace method** (see, for example, [7], [31]). This nullspace characterization of BNP, first reported in James and Plemmons [20], leads to the extension we describe in §5.2.

We begin with a description of the nullspace method itself. Suppose we are given a convenient **particular solution** y_p to the constraint $Ey = b$, and a matrix N whose columns form a basis for the nullspace of E (for convenience, we will call N a **basis matrix**). Then any vector satisfying the constraint can be written as $y = y_p + Nx$ for some choice of x, and minimizing $\|Gy - c\|_2$ subject to the constraint amounts to minimizing $\|G(y_p + Nx) - c\|_2$ over all possible x. The latter minimization is an unconstrained problem of order $n - m_1$. Forming the associated normal equations confirms that the required x solves the symmetric positive definite system

$$N^T G^T G N x = N^T G^T (c - G y_p). \tag{5.2}$$

Now let E be the upper trapezoidal matrix $E_t = \begin{bmatrix} E_L & E_R \end{bmatrix}$ given in (3.6). Note that

$$y_p = \begin{bmatrix} E_L^{-1} b \\ 0 \end{bmatrix} \tag{5.3}$$

is a particular solution of the constraint $E_t y = b$, and the columns of

$$N_I = \begin{bmatrix} -E_L^{-1} E_R \\ I \end{bmatrix} \tag{5.4}$$

form a basis for the nullspace of E_t. Now write the nullspace normal equations (5.2) with these choices of the basis matrix and particular solution:

$$N_I^T G^T G N_I x = N_I^T G^T (c - G y_p). \tag{5.5}$$

Note that we can apply the conjugate gradient algorithm to these normal equations without forming N_I explicitly: an arbitrary matrix-vector product involving N_I requires only a triangular solve with E_L, and a matrix-vector product with E_R. Thus, this approach to solving problem LSE is the simplest possible example of what we call an **implicit nullspace method**, or INM: we solve the nullspace normal equations using a distinguished choice of the basis matrix N, but do not actual form this matrix. In the next section, we develop a more general version of algorithm INM.

Now precondition in the standard way with G_{12}, where G_{12} is the lower righthand corner of the upper triangular matrix G as defined in (3.7):

$$G_{12}^{-T} N_I^T G^T G N_I G_{12}^{-1} w = G_{12}^{-T} N_I^T G^T (c - G y_p), \text{ where } w = G_{12} x. \tag{5.6}$$

Notice that this preconditioned set of normal equations is itself an example of an implicit nullspace method: since N_I is a basis matrix, so is $N_I B$ for any non-singular B. Thus, the linear system (5.6) reflects $N = N_I G_{12}^{-1}$ as the choice of basis matrix. In fact, we can say more about this system. The formidable looking coefficient matrix simplifies nicely, and it will lead directly to the BNP system.

First recall the technique used in §3.1 for augmenting the trapezoidal matrix E_t. The matrix A_1 is defined to be

$$A_1 = \begin{bmatrix} E_L & E_R \\ & G_{12} \end{bmatrix}. \tag{5.7}$$

The inverse of this version of A_1 is given by

$$A_1 = \begin{bmatrix} E_L^{-1} & -E_L^{-1} E_R G_{12}^{-1} \\ & G_{12}^{-1} \end{bmatrix}, \tag{5.8}$$

which in turn tells us that

$$A_1^{-1} \bar{K} = \begin{bmatrix} -E_L^{-1} E_R \\ I \end{bmatrix} G_{12}^{-1} = N_I G_{12}^{-1}. \tag{5.9}$$

Thus $Y = A_2 A_1^{-1} \bar{K}$ can be written

$$Y = G_2 N_I G_{12}^{-1}. \tag{5.10}$$

Now return to the normal equations given in (5.6). From equation (4.16) we know that $G_1 A_1^{-1} \bar{K} = I$. This, combined with (5.10), gives us

$$GN_I G_{12}^{-1} = \begin{bmatrix} G_1 \\ G_2 \end{bmatrix} A_1^{-1} \bar{K} = \begin{bmatrix} I \\ Y \end{bmatrix}. \tag{5.11}$$

Therefore, the coefficient matrix in (5.6) is $I + Y^T Y$, which is precisely the coefficient matrix in BNP.

While the BNP system (3.21) and the preconditioned nullspace system (5.6) involve different unknowns and righthand sides, we can interpret the unknown w in a way which completes the connection between the two methods. Partition

$$y = \begin{bmatrix} y_L \\ y_R \end{bmatrix} \tag{5.12}$$

compatibly with the rows of N_I, and use the fact that $y = y_p + N_I x$ to find that $x = y_R$. Thus, the defining equation $w = G_{12} x$ can be written

$$w = G_{12} y_R = G_1 y. \tag{5.13}$$

But $r_1 = c_1 - G_1 y$ by the definition of r_1. Thus we obtain a relationship between the BNP unknown r_1 and the unknown w in the nullspace normal equations:

$$w = c_1 - r_1. \tag{5.14}$$

Use this fact to rewrite (5.6) as

$$(I + Y^T Y)(c_1 - r_1) = G_{12}^{-T} N_I^T G^T (c - G y_p). \tag{5.15}$$

Finally, move c_1 to the right-hand side and simplify: the result is the BNP system as given in (3.21). Thus algorithm BNP is essentially an implicit nullspace method, with $N = N_I G_{12}^{-1}$ as the (unformed) basis matrix. While we have shown this only for the simple augmentation of the trapezoidal matrix E_t, the conclusion also holds when using interlacing to augment the stairstep matrix E_s.

The only significant distinction between the BNP system (3.21) and the system in (5.15) is the fact that the two systems involve different unknowns (and, of course, different righthand sides). In the next section, we will see that this change in unknown is exactly what we need to produce an algorithm suitable for saddle point problems.

5.2 Algorithm INM: General Case

In the last section, we began with a certain natural basis $N_I = \begin{bmatrix} -E_L^{-1}E_R \\ I \end{bmatrix}$ for the nullspace of the trapezoidal equilibrium matrix E_t, and demonstrated that algorithm BNP can be viewed as an implicit nullspace method with $N = N_I G_{12}^{-1}$ as the basis matrix. We also observed that $N = N_I B$ is a basis matrix for any non-singular matrix B. The basic idea behind the extension of BNP should now be clear: instead of using G_{12}^{-1} as a preconditioner for the nullspace normal equations given in (5.5), precondition (5.5) with any convenient non-singular $(n-m_1)\times(n-m_1)$ matrix B, producing an implicit nullspace method with $N = N_I B$ as the basis matrix. But so far we have only considered the trapezoidal matrix E_t obtained by column pivoting. To make this approach practical, we would like to use interlacing to construct a basis matrix and particular solution for the stairstep matrix E_s.

Recalling equation (3.11), let $P = \begin{bmatrix} P_L & P_R \end{bmatrix}$ be a permutation matrix relating the trapezoidal matrix E_t and the stairstep matrix E_s:

$$E_s P = \begin{bmatrix} E_s P_L & E_s P_R \end{bmatrix} = \begin{bmatrix} E_L & E_R \end{bmatrix} = E_t. \tag{5.16}$$

Now let M_1 be a matrix of size $(n-m_1)\times n$ such that the matrix $\hat{B}_1$ defined by

$$\hat{B}_1 = \begin{bmatrix} E_s \\ M_1 \end{bmatrix} \tag{5.17}$$

is non-singular; we will call such an M_1 an **augmentation matrix.** It is quite easy to construct an M_1 with this property: use rows with leading non-zeros in the $(n-m_1)$ columns in which the stairstep matrix E_s does not have leading non-zeros. Think of M_1 as generalizing the role that G_1 plays in BNP. In the special case $E_s = E_t$, we would have $M_1 = \begin{bmatrix} 0 & M_{12} \end{bmatrix}$, where M_{12} plays the role of G_{12}.

Finally, by analogy with A_1 in equation (3.10), apply interlacing to produce an upper triangular matrix defined by

$$B_1 = P\hat{B}_1 = P\begin{bmatrix} E_s \\ M_1 \end{bmatrix}. \tag{5.18}$$

We are now in a position to define the basis matrix N and the particular solution y_p.

Theorem 5.1 *The vector* $y_p = B_1^{-1}P\begin{bmatrix} b \\ v \end{bmatrix}$ *satisfies the constraint* $E_s y = b$ *for any choice of* v.

Proof. Note that $B_1^{-1} = \hat{B}_1^{-1}P^T$, where $\hat{B}_1$ is defined in equation (5.17). Also, by an argument similar to equation (4.16), we find that $E_s\hat{B}_1^{-1} = \begin{bmatrix} I & 0 \end{bmatrix}$. Use these two facts to establish the result:

$$E_s y_p = E_s\hat{B}_1^{-1}P^T P\begin{bmatrix} b \\ v \end{bmatrix} = E_s\hat{B}_1^{-1}\begin{bmatrix} b \\ v \end{bmatrix} = \begin{bmatrix} I & 0 \end{bmatrix}\begin{bmatrix} b \\ v \end{bmatrix},$$

which is b as required. □

The choices $v = 0$ and $v = c_1$ are particularly convenient when using the results of Theorem 5.1 to define y_p; both are present naturally when initializing quantities in advance of forming B_1.

Theorem 5.2 *The columns of $N = B_1^{-1}P_R$ form a basis for the nullspace of the stairstep matrix E_s.*

Proof. N is the proper size and has full column rank, so we need only establish that $EN = 0$. Recall from the proof of Theorem 5.1 that $B_1^{-1} = \hat{B}_1^{-1}P^T$ and $E_s\hat{B}_1^{-1} = \begin{bmatrix} I & 0 \end{bmatrix}$. Also note that $P^TP_R = \begin{bmatrix} 0 \\ I \end{bmatrix}$ by orthogonality. So

$$EN = E_sB_1^{-1}P_R = E_s\hat{B}_1^{-1}P^TP_R = \begin{bmatrix} I & 0 \end{bmatrix}\begin{bmatrix} 0 \\ I \end{bmatrix},$$

which is zero as required. □

These two theorems give us what we need to describe the implicit nullspace method in a general setting: we implement the algorithm by applying the conjugate gradient method to the factored nullspace normal equations (5.2), with N and y_p as above. Table 5.1 summarizes the algorithm. In the table, s_k is the conjugate gradient direction vector, and $d_k = N^TG^T(c - Gy_p) - N^TG^TGNx^{(k)}$ is the residual associated with the normal equations. The vector q_k stores the product of the coefficient matrix with the direction vector. We use a starting vector of $x^{(0)} = 0$ (which gives us $y^{(0)} = y_p$), but an arbitrary starting vector presents no difficulties.

Unlike BNP, algorithm INM is simple to modify for problems in saddle point form (2.9): when c and G are not available, simply substitute $s = -G^Tc$ in the nullspace normal equations (5.2) to obtain the new righthand side

$$h = -N^T(s + Fy_p). \tag{5.19}$$

Also note that it is easy to preserve the opportunities for block-based parallelism discussed in §3.1: given a substructured problem, we need only construct M_1

Table 5.1: Implicit Nullspace Method (General Case)

1. Use Gauss Elimination or orthogonal reduction on E and b to replace E with its stairstep form E_s.
2. Choose a convenient augmentation matrix M_1. The interlacing information (represented by the permutation matrix P below) can be stored in a pointer vector.
3. Form $B_1 = P\begin{bmatrix} E_s \\ M_1 \end{bmatrix}$ and $b_0 = P\begin{bmatrix} b \\ 0 \end{bmatrix}$.
4. Initialize:
   (a) $x^{(0)} = 0$
   (b) $d_0 = P_R^T B_1^{-T} G^T(c - GB_1^{-1}b_0)$
   (c) $s_0 = d_0$
5. For $k = 0, 1, \ldots,$ until $d_k^T d_k <$ tolerance:
   (a) $q_k = P_R^T B_1^{-T} F B_1^{-1} P_R s_k$
   (b) $\alpha_k = d_k^T d_k / s_k^T q_k$
   (c) $\begin{bmatrix} x^{(k+1)} \\ d_{k+1} \end{bmatrix} = \begin{bmatrix} x^{(k)} \\ d_k \end{bmatrix} + \alpha_k \begin{bmatrix} s_k \\ -q_k \end{bmatrix}$
   (d) $\beta_{k+1} = d_{k+1}^T d_{k+1} / d_k^T d_k$
   (e) $s_{k+1} = d_{k+1} + \beta_{k+1} s_k$
6. Recover $y = B_1^{-1}(P_R x + b_0)$ and exit.

so that it conforms to the substructuring to ensure that interlacing produces an upper triangular matrix with the proper structure.

It remains to show that we can deal with problems in which G lacks full column rank. We will deal with this topic in the next section, when we look at specific choices of the augmentation matrix M_1.

5.3 Examples of Implicit Nullspace Methods

In this section, we propose some specific examples of implicit nullspace methods, and discuss circumstances under which they might be appropriate. Each is based on a natural choice of the augmentation matrix M_1.

Algorithm INMI

The simplest possible implicit nullspace method, which we call algorithm INMI, involves augmenting the stairstep matrix E_s with rows of the (scaled or unscaled) identity matrix I. If, for example, the matrix E_s has the pattern

$$E_s = \begin{bmatrix} & * & & * & * & & \\ & & & * & & & * \\ & & & & * & * & \\ & & & & & & * \end{bmatrix}, \tag{5.20}$$

then define

$$M_1 = \begin{bmatrix} 1 & & & & & & \\ & & 1 & & & & \\ & & & & & 1 & \end{bmatrix}. \tag{5.21}$$

After interlacing, we have

$$B_1 = \begin{bmatrix} 1 & & & & & & \\ & * & & * & * & & \\ & & 1 & & & & \\ & & & * & & & * \\ & & & & * & * & \\ & & & & & 1 & \\ & & & & & & * \end{bmatrix}. \tag{5.22}$$

If E_s has substructured form, then so does B_1, and triangular solves involving B_1 and B_1^T may be done in parallel as described in §3.1. Moreover, these solves can be coded to exploit the presence of rows of the identity.

A formal description of the INMI interlacing begins with a permutation matrix P relating the stairstep form E_s to the trapezoidal form E_t. Given

$$E_s P = \begin{bmatrix} E_s P_L & E_s P_R \end{bmatrix} = \begin{bmatrix} E_L & E_R \end{bmatrix} = E_t \tag{5.23}$$

as in (3.11), the matrix $M_1 = P_R^T$ is precisely the augmentation matrix we seek. With this choice of M_1, it is easy to show that the basis matrix $N = B_1^{-1} P_R$ reduces to $N = P N_I$, where N_I is the fundamental basis matrix defined in equation (5.4). Similarly, the particular solution $y_p = B_1^{-1} P \begin{bmatrix} b \\ 0 \end{bmatrix}$ is actually $y_p = P \begin{bmatrix} E_L^{-1} b \\ 0 \end{bmatrix}$, which is a row permutation of the particular solution given in (5.3). Thus INMI is nothing but the implicit nullspace method in (5.5), corrected for interlacing.

Algorithm INMI is one way to solve problems in which G is unavailable, prohibitively dense, lacks full column rank, or is otherwise unsuitable for interlacing; in particular, INMI is capable of handling saddle point problems. But, since we ignore all information in the matrix G when constructing the preconditioner, there is good reason to expect little preconditioning effect beyond the order reduction itself. This is in fact what we observed in our experiments (see chapter 6).

Algorithm INMG

If G has full column rank and convenient structure, we can augment E_s by interlacing rows of G, exactly as we do in algorithm BNP. In fact, as shown in the previous section, the resulting implicit nullspace method, which we call algorithm INMG, has the same coefficient matrix as the BNP system. Of course, as with BNP, algorithm INMG may fail if G lacks full column rank, since the rows of G needed to augment E_s may not be available.

The linear systems in BNP and INMG differ only in their unknowns and righthand sides. Let x be the unknown coordinate vector in INMG (in §5.2 we used w for other reasons). Then, by the analysis in the previous section, x is related to the unknown r_1 in BNP by

$$x = c_1 - r_1. \tag{5.24}$$

Recall from §5.2 that the translated unknown in INMG gives us an important advantage over algorithm BNP: unlike BNP, algorithm INMG can be used to solve saddle point problems. We demonstrate this on a Stokes problem in chapter 6.

In the special case $c = 0$ (e.g. the structures application), the two unknowns satisfy $x = -r_1$, so the BNP and INMG systems differ only by a negative sign. The stop criteria in the algorithms also have the same interpretation, so we should expect the two algorithms to perform identically. In fact, INMG outperformed BNP in all our experiments (see chapter 6); this is how we identified the instability in the BNP recursion (3.34) defining the defect ν_k.

As with each of the methods we describe, there are opportunities to code solves and matrix-vector products in INMG to exploit the special properties of the preconditioner. In particular, depending on the structure of F and G, one may want to use the fact that $G_1 B_1^{-1} P_R = I$ to compute the conjugate gradient direction vector s_k.

$$E_s = \begin{bmatrix} & * & & * & * & & \\ & & & * & & & * \\ & & & & * & * & \\ & & & & & & * \end{bmatrix}$$

$$B_1 = \begin{bmatrix} \bullet & \bullet & \bullet & & & & \\ & * & & * & * & & \\ & & 1 & & & & \\ & & & * & & & * \\ & & & & * & * & \\ & & & & & \bullet & \bullet \\ & & & & & & * \end{bmatrix}$$

$$G = \begin{bmatrix} \bullet & \bullet & \bullet & & & & \\ & \bullet & \bullet & & & & \\ & & & \bullet & \bullet & & \\ & & & & \bullet & & \\ & & & & & \bullet & \bullet \\ & & & & & & \bullet \end{bmatrix}$$

Figure 5.1: Interlacing in Algorithm INMGI

Algorithm INMGI

We have already observed several times that if G lacks full column rank, we may be unable to interlace rows of E_s and G to produce an upper triangular matrix B_1. In particular, if there is no row of E_s with leading non-zero in column j, then interlacing requires the existence of a row of G with leading non-zero in that column. When G lacks full column rank, such a row of G may or may not exist.

There is, however, an obvious way to combine the techniques in algorithms INMI and INMG to overcome this difficulty. One can use rows of G to augment E_s when such rows are available, and use rows of the identity matrix I when there is no appropriate row in G. We call this approach algorithm INMGI.

Consider, for example, the matrices in figure 5.1. To produce B_1, the augmentation matrix M_1 requires rows with leading non-zeros in each of columns 1, 3, and 6. There are rows of G with leading non-zeros in columns 1 and 6, so we include these rows in M_1. The remaining row of M_1 comes from the identity matrix; after interlacing, B_1 has the depicted form.

While this approach is successful in solving problems involving rank deficient G, there is a potential problem: unless the non-zeros in *all* rows of M_1 are of roughly the same magnitude, the resulting basis matrix may have columns of widely varying size, resulting in a badly conditioned coefficient matrix. Consider a very simple example. Let E be of the form

$$E = \begin{bmatrix} E_L & E_C & E_R \end{bmatrix}, \tag{5.25}$$

where E_L is upper triangular and non-singular. Suppose there are no rows of G with leading non-zeros in the columns associated with E_C. Then form

$$B_1 = \begin{bmatrix} E_L & E_C & E_R \\ & D & \\ & & G_{12} \end{bmatrix}, \tag{5.26}$$

where D is a diagonal matrix (a scaled version of the portion of I used in the augmentation), and G_{12} is again the lower righthand corner of G. We can explicitly calculate the inverse of B_1; from that, we obtain the basis matrix:

$$N = \begin{bmatrix} C_1 D^{-1} & C_2 G_{12}^{-1} \end{bmatrix}, \tag{5.27}$$

where C_1 and C_2 are given by

$$C_1 = \begin{bmatrix} -E_L^{-1} E_C \\ I \\ 0 \end{bmatrix}, \quad C_2 = \begin{bmatrix} -E_L^{-1} E_R \\ 0 \\ I \end{bmatrix}. \tag{5.28}$$

Assuming the non-zero elements of E are of roughly the same magnitude, the non-zeros of C_1 and C_2 are likely to be of order one. Thus, the conditioning of the basis matrix depends on D_1^{-1} and G_{12}^{-1}. If the non-zeros in one of these matrices are of different magnitude than those in the other, the basis matrix (and hence the nullspace normal equations) will probably be poorly conditioned.

We observed this phenonenon in our experiments: without scaling, convergence of INMGI actually required more iterations than the size of the problem. In our engineering test problems, however, the blocks of G were all of comparable magnitude. Thus, we simply scaled G and c by a constant multiple of the identity so that the non-zeros in G were $O(1)$. The results, reported in chapter 6, were much more encouraging.

Algorithm INMF

We have already proposed one implicit nullspace method, algorithm INMI, capable of solving problems in which the matrix G is either unavailable or unsuitable for interlacing. Intuitively, though, this approach seems less than promising, since we ignore all information in G when constructing the preconditioner. Experiments reported in chapter 6 confirm that the preconditioner produced by INMI is not effective at all.

There is, however, another way to approach the problem: we can use information in $F = G^T G$ to construct the augmentation matrix. One possibility is to generate a Cholesky factorization of a portion of F (for example, the block diagonal portion), then use rows of this factorization to form M_1. Another option, as yet untested, is to use selected rows of an **incomplete Cholesky factorization** of F (see the appendix for a brief description of such a factorization). In both cases, the intent is that the factorization produces a $\hat{G}$ which is at least a rough approximation of the matrix G.

While intuition suggests that this approach should produce a better preconditioner than algorithm INMI, there is one potential problem with using approximate factorizations of F to form M_1. It is important to remember that we are not using the entire factorization as a preconditioner; we are instead using only *selected rows* of what is already a crude approximation to G. This may result in an augmentation matrix of little or no value. It is possible, perhaps likely, that the cost of both producing the factorization and using the resulting B_1 within iterations may prove to be wasted effort. In fact, this is precisely what we observe in one of the experiments described in the next chapter.

In chapter 6, we report on experiments with most of the methods described here. We note, however, that there are other possibilities which remain untested. One could, for example, use an **incomplete orthogonal factorization** of a complicated matrix G to produce a simpler approximation $\hat{G}$ suitable for interlacing (see appendix). Another possibility is what one might call a **partial orthogonal factorization** of G: instead of reducing G completely, use orthogonal rotations on only a selected portion of G to produce a matrix $\hat{G}$ suitable for interlacing. It is clear that the possibilities are endless: algorithm INM offers a great deal of flexibility in constructing the preconditioner. What we have yet to establish is whether any of the proposed algorithms efficiently solve problem LSE in its various formulations.

6. Numerical Experiments

In this chapter, we report on the results of numerical experiments with the iterative algorithms discussed in this thesis, including both the conjugate gradient methods (BNP, Pwgt, and various forms of INM) and the linear stationary methods (2-cyclic SOR and 3-AOR). We compare and contrast the behavior of the algorithms, and examine parallel performance on substructured problems.

Section 6.1 is a summary of the conditions under which we conducted the tests. In §6.2, we consider three small models of elastic structures. Besides providing us with an opportunity to validate the codes, these problems offer the best means of studying the behavior of 2-SOR and 3-AOR. Then, in §6.3, we consider the performance of the conjugate gradient methods on elastic problems of more realistic size.

In the last two sections, we look carefully at the extensions to algorithm BNP by considering problems for which BNP is not well suited. Section 6.4 describes models of structures with simulated "damage," producing problems in which the matrix G lacks full column rank. In §6.5 we consider the marker-and-cell saddle point formulation of the Stokes problem.

6.1 Overview of Experiments

All experiments were run on a two-processor Alliant FX/40. We employed full optimization, including vectorization, on all test problems, but we inhibited vectorization within the codes when it was appropriate to do so. We also used the DAS compiler option, allowing the compiler to assume that finite precision arithmetic is associative. This gives the compiler the flexibility to rearrange the order of computations, enhancing the opportunity for concurrent execution. Occasionally, however, the DAS option causes the results for an experiment run on two processors to differ slightly from those obtained on a single processor; in particular, one sometimes observes minor differences in the total number of iterations required to achieve a solution of a specified accuracy. To simplify the tables, we report iteration counts only for the two processor case.

Execution times (in seconds), obtained using the Alliant `etime` intrinsic, include all operations except input/output (for the Stokes problems, this includes the cost of generating the matrices). In all problems, however, only the iteration times were significant: pre-processing, including factoring E and completing the interlacing step, typically required only 1–2% of the cpu time.

For convenience, we list the algorithms tested in our experiments:

BNP: order-reducing conjugate gradient method due to Barlow, Nichols, and Plemmons (§3.2).

Pwgt: preconditioned weighting method (§3.3).

2-SOR: optimal 2-cyclic successive over-relaxation applied to the modified Kuhn-Tucker equations (§3.3).

3-AOR: optimal three-block accelerated over-relaxation applied to the modified Kuhn-Tucker equations (§3.3).

INMI: implicit nullspace method in which the preconditioner is formed by interlacing rows of E with rows of the identity matrix I (§5.3).

INMG: implicit nullspace method in which the preconditioner is formed by interlacing rows of E with rows of G (§5.3); essentially equivalent to algorithm BNP.

INMGI: implicit nullspace method in which the preconditioner is formed by interlacing rows of E with rows of G when the appropriate rows of G are available. When such rows are not available, rows of the identity matrix I are used to complete the construction (§5.3).

INMF: implicit nullspace method in which the preconditioner is formed by interlacing rows of E with rows constructed in some way from information in the matrix F (§5.3). In our tests on the Stokes problem, we use rows from the Cholesky factor of the block diagonal portion of F.

All programs were written in FORTRAN77 using double precision arithmetic and sparse, row-oriented data structures. Since we attempted to solve three distinct types of problems (elastic structures, structures with rank deficient blocks in G, and Stokes flow), we coded as many as three versions of a given algorithm. For each type of problem, we designed the codes so that the data structures, logic, and primary subroutines (especially the factorization of E, the triangular solves, and matrix-vector multiplications) were as similar as possible from one algorithm to another. We did, of course, try to take advantage of special features of each algorithm. Thus, for example, the codes for algorithm INMI accomplish triangular solves involving B_1 by exploiting the fact that B_1 contains rows of the identity matrix.

We did not write special codes for algorithm INMG. Instead, we used the INMGI codes to run algorithm INMG on full rank problems (algorithm INMGI is equivalent to INMG in this case, since the rows of G needed for interlacing are always available). The logic and data structures in INMGI are somewhat more complex than for the other algorithms, since the code must be able to identify and deal with column rank deficiencies in G. Thus, the times reported for algorithm INMG are slightly slower than they would be had we written code specifically designed for full rank problems.

We decided that the fairest, most meaningful way to compare the algorithms was to report the times required to produce a result of specified accuracy. Therefore, for each problem we constructed a "true" solution (by means described in each section), and adjusted the stop tolerance for each algorithm until the reported error (in the infinity norm) was as close as possible to a specified accuracy. We used the standard stop criterion for all conjugate gradient methods: if the algorithm solves the symmetric positive definite system $Kz = f$, terminate execution when the quantity $\|f - Kz^{(k)}\|_2^2$ is less than the specified tolerance (this quantity occurs naturally within each iteration). For the linear stationary methods, we terminated execution when the change in successive iterates was smaller than the specified tolerance.

In the preconditioned weighting algorithm, we experimented with the value of the weighting parameter τ to obtain the best possible results. In all cases, the best choice of τ was between 10^5 and 10^7, with little difference in performance as τ varied over this range. This is consistent with the theory in Barlow [2].

We also report that we have have run the codes on several other architectures, including the Cray Y-MP, the Alliant FX/8, and Sun 3/50 workstations. We have only anecdotal results on these machines, and did not attempt to optimize the ported codes. Therefore, we include here only results for experiments on the FX/40.

6.2 Small Full-Rank Structures Problems

The three small test problems described in this section are all models of elastic structures (see §2.2). The resulting constrained minimization problems involve a symmetric positive definite matrix F which is block diagonal. Thus, the corresponding Cholesky factor G is square, of full rank, and block diagonal with upper triangular blocks. We used these problems as a means of validating our codes, but they also gave us a chance to compare the linear stationary methods (2-SOR and 3-AOR) against the conjugate gradient algorithms (BNP, INMG, INMI, and Pwgt).

Problem **WRENCH4** (figure 6.1a) is a substructured version of a test problem due to Lawo [6]. It consists of 48 planar elements, and produces a problem with 112 constraints and 216 unknowns. The diagonal blocks in the element flexibility matrix are 5×5 for the rectangular elements, and 3×3 for the triangular elements. The applied force is indicated by the arrows in the figure. We partitioned the structure into four substructures as shown in the figure; the resulting transition zone has 50 columns.

Problem **DAM2** (figure 6.1b) is a trapezoidal region intended to be a rough approximation of a cross-section of a dam, modelled using square and triangular

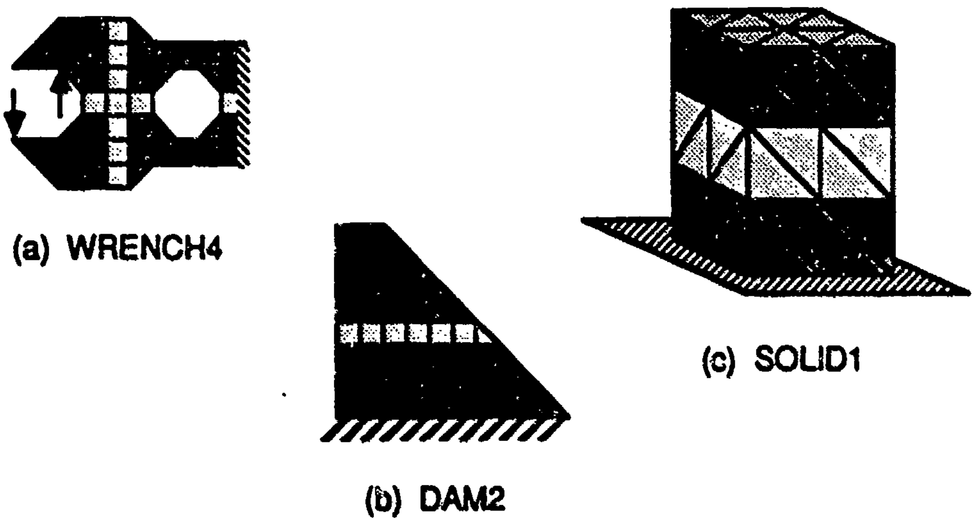


Figure 6.1: Models for Small Full-Rank Problems

planar elements as described in Przemieniecki [32]. Like WRENCH4, the blocks in F are 5×5 for the rectangular elements, and 3×3 for the triangular elements. The external load simulates a body of water against the left vertical wall. The model produces a problem with 104 constraints and 244 unknowns. There are two substructures as shown in the figure; the resulting transition zone has 33 columns.

Problem SOLID1 (figure 6.1c), also modelled using techniques in [32], approximates a building subjected to the force of a steady wind approaching one of its vertical edges. This version consists of 60 solid tetrahedral elements (five tetrahedrons in each small cube), and produces a problem with 81 constraints and 360 unknowns. Each of the 60 blocks in the block diagonal matrix F is 6×6. Again there are two substructures. The resulting transition zone is quite large, consisting of 120 out of 360 columns.

In practice, these problems are all too small to justify substructuring techniques; the transition zones are far too large, and the substructures themselves are not well balanced. We constructed substructured versions of the small problems primarily to validate the codes. Note that we deliberately chose partitionings which produced at least one unstable substructure (see §2.4) so that there would be diagonal blocks in E which were deficient in row rank. In all three problems, we could have chosen natural partitionings which produce stable, full rank substructures.

To measure error, we obtained the "true" solution to each problem by solving the original Kuhn-Tucker equations (2.6) using LINPACK [8]. We then adjusted the stop tolerances for each algorithm so that the infinity norm of the reported error was roughly 10^{-4}. We summarize the results of the experiments in table 6.1.

The results for SOR and AOR are for the approximate optimal values of the iteration parameters (obtained experimentally). We report that AOR is highly sensitive to the choice of the parameters: very small deviations from the optimal values result in either divergence or a drastic reduction in the rate of convergence. In all three test problems, the region of convergence in the ω–β plane appears to be a tiny, narrow crescent-shaped region. The choice of parameter in 2-cyclic SOR is consistent with the theory (see [4] and [24]): convergence occurs for all ω below a sufficiently small critical value, with the optimal choice occurring very close to that critical value. It is clear from the tables that these test problems do not satisfy the conditions in Papadopoulou, Saridakis, and Papatheodorou [28], since 2-SOR outperforms 3-AOR by a wide margin. Moreover, for each of these problems, all of the conjugate gradient algorithms proved vastly superior to the linear stationary methods.

Table 6.1: Numerical Results: Small Full-Rank Structures Problems

WRENCH4
(216 unknowns, 112 constraints)

	BNP	INMG	PWgt	INMI	2-SOR	3-AOR
Iterations*	33	33	34	40	457	2439
2 proc time	.345	.352	.343	.355	3.58	19.0
1 proc time	.535	.550	.522	.618	5.75	30.7
Speedup	1.55	1.56	1.52	1.74	1.61	1.62

DAM2
(244 unknowns, 104 constraints)

	BNP	INMG	PWgt	INMI	2-SOR	3-AOR
Iterations*	52	49	51	78	818	5991
2 proc time	.595	.609	.571	.791	8.24	60.8
1 proc time	.899	.911	.852	1.30	12.4	91.7
Speedup	1.51	1.50	1.49	1.64	1.50	1.51

SOLID1
(360 unknowns, 81 constraints)

	BNP	INMG	PWgt	INMI	2-SOR	3-AOR
Iterations*	41	41	42	88	208	609
2 proc time	.735	.773	.721	1.32	3.32	9.56
1 proc time	1.08	1.13	1.06	2.06	4.81	14.0
Speedup	1.47	1.46	1.47	1.56	1.45	1.46

*Minor differences occasionally occur when changing number of processors. Statistics are for two-processor runs.

Algorithms BNP, INMG, and PWgt all performed comparably on these problems. This is as expected: BNP and INMG are solving the same linear system (§5.3), and both BNP and INMG are essentially the limiting case of PWgt (§4.3). In the next section, however, we will observe that these algorithms behave differently on larger test problems.

Note that INMI performed measurably slower than the other conjugate gradient algorithms, especially on the three-dimensional problem SOLID1. This is consistent with our expectations: while INMI is an order-reducing method, the algorithm ignores all information in the matrix G when constructing the preconditioner.

On all problems, speedups are quite reasonable given the large transition time and imbalances in the substructures. They are also consistent across the algorithms; this reflects the fact that the principal subroutines are similar in each of the codes.

6.3 Larger Full-Rank Structures Problems

Having validated the codes on the small test problems described in the previous section, we then experimented with larger elastic structures problems; we report the results of those tests here. Because of the poor performance of 2-SOR and 3-AOR on the small test problems, and the difficulty of determining appropriate iteration parameters for these algorithms, we did not test the linear stationary methods on this set of problems.

Problem DAM10 (figure 6.2a) is similar to DAM2 (see §6.2), except that there are more elements in the model. There are 1,220 planar elements, producing a problem with 2,440 constraints and 6,020 unknowns. We consider two

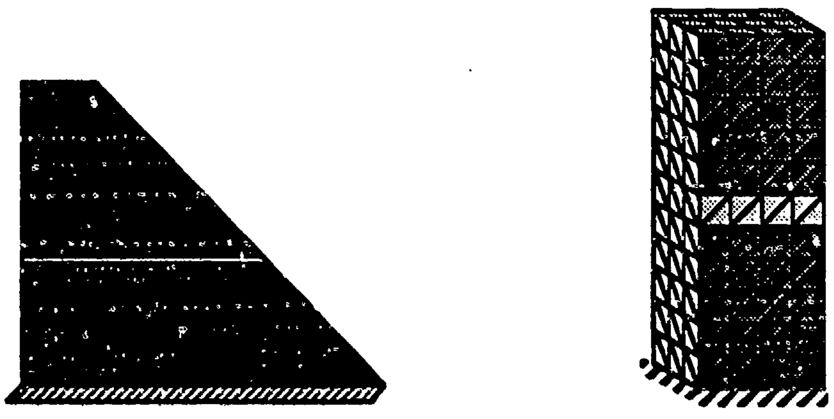


Figure 6.2: Models for Large Full-Rank Problems

versions of the problem: one involving no substructuring of the physical domain, and another with two substructures of fairly equal size (and 178 columns in the transition zone). Again, we deliberately chose a transition zone consisting of a horizontal strip through the domain (see the figure), so that the upper substructure would be unstable, and one of the diagonal blocks in the equilibrium matrix would lack full row rank.

Problem SOLID2 (figure 6.2b) is similar to SOLID1 (see §6.2), except that the rectangular solid is very tall, and there are more elements in the model. There are 220 free nodes and 660 tetrahedral elements in this model, producing a problem with 660 constraints and 3,960 unknowns. As with DAM10, we consider one version with no substructuring, and a second version with two substructures. Even though the matrices are fairly large for this problem, the geometry of the model does not allow a small transition zone: there are 360 transition columns, which is almost 10% of the total number of columns in the equilibrium matrix.

As with the problems in the previous section, we compared the solutions produced by each algorithm to a reference vector assumed to be the "true" solution. This time, however, the problems were too large to obtain a solution using LINPACK on the Kuhn-Tucker equations. Since all algorithms produced results consistent with the LINPACK solutions on the smaller versions of the structures problems, we felt confident that the codes were working correctly. Therefore, we solved each problem using algorithm INMG with a stop tolerance of $\epsilon = 10^{-10}$, and used the resulting solution as the "true" solution when computing errors. We then adjusted the stop tolerances on each algorithm so the reported error (in the infinity norm) was roughly 2×10^{-4}.

We summarize the results of the experiments in table 6.2. Perhaps the most interesting difference between these results and those in table 6.1 concerns the relative performance of BNP and INMG. Despite the fact that the two methods solve the same linear system in essentially the same way, algorithm INMG is superior, especially on DAM10. Here we see clearly for the first time that the method used to calculate the defect in algorithm BNP is unstable: it causes inaccuracies in the direction vectors and scale factors, resulting in slower convergence (see §3.2). If we replace the non-standard calculation of the defect with the more conventional update based on explicit use of a direction vector and scale factor, the results for BNP coincide almost exactly with INMG. In this case, the two codes are line-for-line virtually identical; the major difference is the fact that algorithm BNP updates the original unknown y at each iteration, while INMG recovers y after iterations cease (§5.3).

If we think of algorithm INMG as an improved version of BNP, the results for PWgt make sense. Since BNP (and therefore INMG) is the limiting case

Table 6.2: Numerical Results: Large Full-Rank Structures Problems

DAM10
(6,020 unknowns, 2,440 constraints)

Two Substructures:

	BNP	INMG	PWgt	INMI
Iterations*	1148	910	955	1416
2 proc time	282.	227.	232.	355.
1 proc time	483.	386.	399.	603.
Speedup	1.71	1.70	1.72	1.70

No Substructuring:

	BNP	INMG	PWgt	INMI
Iterations	937	782.	830	1139
1 proc time	354.	304.	312.	444.

SOLID2
(3,960 unknowns, 660 constraints)

Two Substructures:

	BNP	INMG	PWgt	INMI
Iterations*	231	223	227	689
2 proc time	64.2	63.3	62.1	191.
1 proc time	86.4	87.5	87.3	277.
Speedup	1.35	1.38	1.41	1.45

No Substructuring:

	BNP	INMG	PWgt	INMI
Iterations	453	429	432	1146
1 proc time	114.	112.	103.	291.

*Minor differences occasionally occur when changing number of processors. Statistics are for two-processor runs.

of the preconditioned weighting algorithm, it is only logical that the results for INMG are slightly better than those for PWgt. Because BNP in its original form is converging more slowly than it ought to (see previous paragraph), algorithm PWgt outperforms BNP on both problems.

Comparing the results with and without substructuring is also quite interesting: the problems change character completely when one changes the partitioning of the physical domain. For DAM10, convergence is significantly better without substructuring, while for SOLID2, it it significantly better with two substructures. On both problems, however, the results with two substructures on two processors are superior to those obtained with one processor on one substructure. Speedups (comparing substructured results on one versus two processors) were not as good as we would have liked, but clearly reflect the size of the transition zone: tests on SOLID2, which has a large transition zone, exhibit significantly poorer parallel performance.

Once again, INMI performed badly. This was true even when we experimented with various types of scaling. It is quite clear that one cannot afford to ignore the matrix G in constructing a preconditioner for an implicit nullspace method.

6.4 Rank-Deficient Structures Problems

To test our ability to solve problem LSE when the matrix G lacks full column rank, we modified the problems described in the previous section to simulate the presence of "damaged" elements. We did this by defining a "damaged" region in each model, and setting Poisson's ratio ν to the appropriate critical value for each element in the region (see §2.2). The associated blocks of the matrix F

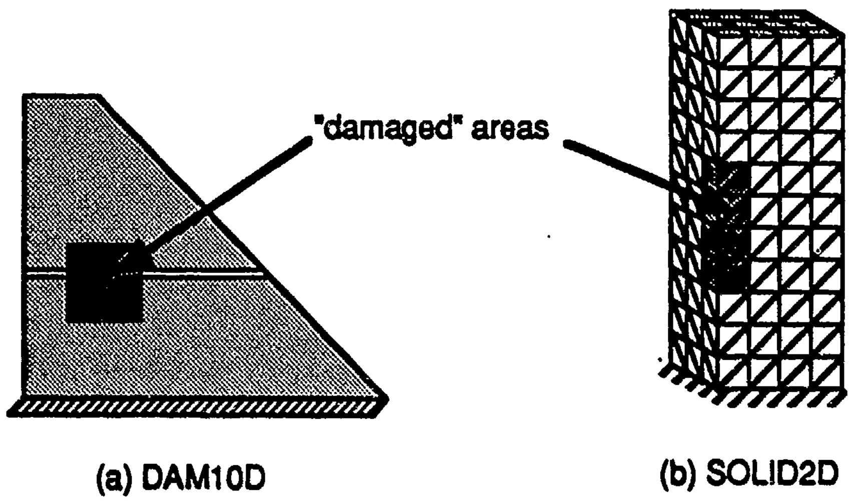


Figure 6.3: Models for Rank Deficient Problems

are then non-negative definite and singular, so the corresponding blocks in the generalized Cholesky factor F are rank deficient. The modified regions in each problem are small enough to ensure that hypothesis H2 (§2.1) is still satisfied, so the problems remain well posed.

For the reasons described in chapter 5, interlacing with rows of G fails for these problems. Thus, algorithms BNP and Pwgt, at least as we have implemented them, cannot be used. The same is true of algorithm INMG, which is essentially an improved version of BNP. We are left with algorithms INMCI and INMI.

Problem **DAM10D** (figure 6.3a) is a modified version of DAM10 (see §6.3); there are 2,440 constraints and 6,020 unknowns. The modified region (the dark area in the figure), consists of 100 elements, or 8% of the total area of the model. The corresponding blocks in F and G are 5×5 with rank 4. Thus the rank of G is 5,920; this is 100 short of the total number of columns in G. Despite a rank deficiency of 100, only 10 rows of the scaled identity matrix were needed by the INMGI to produce an upper triangular matrix B_1.

Problem SOLID2D (figure 6.3b) is a modified version of SOLID2 (see §6.3); there are 660 constraints and 3,960 unknowns. The modified region (the dark area in the figure), consists of 20 elements, or 3% of the total volume of the model. The corresponding blocks in F and G are 6×6 with rank 5. Thus the rank of G is 3,940; this is 20 short of the total number of columns in G. Algorithm INMGI required 12 rows of the scaled identity matrix to complete the construction of B_1.

Once again, these problems are too large to obtain a solution by using LINPACK on the Kuhn-Tucker equations. We obtained solutions consistent with the LINPACK solutions on smaller versions of these problems, so we felt confident that the codes were working correctly. Therefore, we solved each problem using algorithm INMGI with a stop tolerance of $\epsilon = 10^{-9}$, and used the resulting solution as the "true" solution when computing errors. We then adjusted the stop tolerances on each algorithm so the reported error (in the infinity norm) was roughly 10^{-4}.

Table 6.3 is a summary of the results for these test problems. While both INMGI and INMI successfully solve the problems, algorithm INMGI is clearly superior. Algorithm INMGI uses as much information as possible from the matrix G. In both problems, very few rows of the identity matrix are needed to form B_1, so algorithm INMGI is "almost" INMG for these problems. We do, however, note that it is necessary to scale the matrices as described in §5.3 to achieve these results: when the rows of G are not $O(1)$ (or, equivalently, if we do not scale the identity matrix before using it for interlacing), the performance

Table 6.3: Numerical Results: Rank Deficient Structures Problems

DAM19D
6,020 unknowns, 2,440 constraints
Rank of G: 5,920

Two Substructures:

	INMGI	INMI
Iterations*	1645	2380
2 proc time	410.	591.
1 proc time	713.	1020.
Speedup	1.74	1.73

SOLID2D
(3,960 unknowns, 660 constraints)
Rank of G: 3,940

Two Substructures:

	INMGI	INMI
Iterations*	1266	1802
2 proc time	349.	497.
1 proc time	478.	692.
Speedup	1.37	1.39

*Minor differences occasionally occur when changing number of processors. Statistics are for two-processor runs.

is INMGI is unacceptable.

6.5 Stokes Flow

To test our ability to solve saddle point problems, we experimented with the marker-and-cell formulation of Stokes flow on the unit square (§2.3). As mentioned in chapter 4, algorithm BNP cannot be used to solve this problem. Algorithm INMG, which is the nullspace version of BNP, can deal with it easily. We can also use Pwgt, INMI, and any of several versions of algorithm INMF. To test algorithm INMF, we used selected rows of the Cholesky factor of the block diagonal part of F to augment the equilibrium matrix. Recalling from §2.3 that the diagonal blocks of F are tridiagonal, this means the code used selected rows of the Cholesky factor of the tridiagonal part of F to form the augmentation matrix. We did not form F explicitly in any of the codes; the subroutine which computes the matrix-vector product Fx in INMG, INMF, and INMI uses the stencil defining the action of F on an arbitrary vector (again, see §2.3). This routine is quite fast, and explains the fact that the time required for each iteration of these algorithms is about half that required for an iteration of Pwgt.

We tested both a 50×50 grid (resulting in a problem with 2,499 pressure constraints and 4,900 unknown velocity components), and a 100×100 grid (9,999 pressure constraints and 19,800 unknowns). We ran each problem with and without substructuring; there were two sub-domains in the substructured versions. Algorithms INMG and PWgt, which require that the matrix G reflect the substructuring, employ the wide transition zone described in §2.4. Algorithms INMI and INMF do not use G; we therefore used the narrow transition zone described in that section. In both cases, the transition zone constitutes a small percentage of the columns of E.

We used the following artificial "flow" as our test problem:

$$\begin{aligned} v_1 &= 2x\cos y \\ v_2 &= -x^2\sin y \\ p &= xy^2. \end{aligned}$$

Here v_1 and v_2 are the horizontal and vertical components of the (continuous) velocity vector, and p represents pressure. The velocity has non-zero divergence, but is irrotational (curl $v = 0$). In fairness, we report that we have been unable to use our codes to solve for rotational flows: on several problems involving non-zero curl, we obtained errors (based on the true continuous solution) on the order of 2×10^{-3} regardless of the specified mesh. We have not been able to determine the cause of this behavior.

We used the continuous solution as given above to measure the error obtained by each algorithm. We varied the stop tolerances for each algorithm to determine the smallest attainable error on the given grid, and then ran the tests of each algorithm with the largest stop tolerance which produced this error. For the 50×50 grid, the smallest attainable error (measured in the infinity norm) was roughly 6×10^{-5} for all algorithms; for the 100×100 grid, the best attainable error was roughly 3×10^{-5}. Thus, the discretization appears to be an $O(h)$ global approximation to the continuous problem.

Table 6.4 summarizes the results of the experiments. We observe that the tri-diagonal portion of F produces a poor preconditioner in INMF. It seems we cannot afford to ignore the information in the outer bands of F when construct-

Table 6.4: Numerical Results: Stokes Flow on Unit Square

50×50 Grid
(4,900 unknowns, 2,499 constraints)

Two Substructures:

	INMG	PWgt	INMI	INMF
Iterations*	354	433	1601	1637
2 proc time	69.4	85.6	127.	157.
1 proc time	127.	157.	224.	283.
Speedup	1.83	1.83	1.76	1.80

No Substructuring:

	INMG	PWgt	INMI	INMF
Iterations	154	234	1605	1681
1 proc time	26.1	85.2	214.	272.

100×100 Grid
(19,800 unknowns, 9,999 constraints)

Two Substructures:

	INMG	PWgt	INMI	INMF
Iterations*	1132	1370	8534	8352
2 proc time	463.	1120.	2990.	3400.
1 proc time	835.	2120.	5180.	6090.
Speedup	1.80	1.89	1.73	1.79

No Substructuring:

	INMG	PWgt	INMI	INMF
Iterations	385	594	9721	8503
1 proc time	274.	886.	4990.	5930.

*Minor differences occasionally occur when changing number of processors. Statistics are for two-processor runs.

ing the preconditioner, especially when we are only using selected rows of the factorization we obtain. INMI also performs poorly, as it did on all other test problems.

Algorithm INMG, on the other hand, performs fairly well; the natural factor G of the Laplacian matrix F (see §2.3) allows us to take advantage of some outer band information in constructing the preconditioner. The difference in iterations between INMG and PWgt is greater here than it is for any other class of problems; the substantial difference in total time is due to INMG's efficient matrix-vector multiplier (based on the stencil defining F).

Notice we obtain quite reasonable speedups on the Stokes problem: the transition zones are fairly small, and the subdomains have virtually identical structure. On the other hand, the results for INMG without domain decomposition reflect an extraordinary reduction in the number of iterations. The change in the rate of convergence is so dramatic that the time required to solve the problem on one processor without domain decomposition is almost half that needed for the substructured problem on two processors. Clearly one would need a more sophisticated approach to constructing M_1 to overcome the penalty associated with domain decomposition applied to the Stokes problem. Incomplete Cholesky decompositions offer one possibility worth exploring.

We note that we made no attempt to exploit the underlying continuous problem to construct a plausible starting vector for the iterations. We were interested in the discretized Stokes problem only as an example of a linear system in saddle point form. The experiments indicate that we have in fact succeeded in extending BNP to a class of algorithms capable of solving such systems.

7. Conclusions

The analysis and experiments described in this thesis hardly constitute an exhaustive study of the subject at hand. But the work does seem to make a convincing case that algorithm BNP in particular, and order-reducing conjugate gradient methods in general, offer a promising alternative to existing methods for solving large, sparse constrained minimization problems.

The proofs that algorithm BNP is superior to p-cyclic SOR [4] and block AOR (chapter 4) hold only in exact arithmetic. But the results are decisive for the test problems we have considered: even the poorest implicit nullspace method proved faster than optimal 2-SOR and 3-AOR by a wide margin. Additionally, order-reducing conjugate gradient methods free the user of the burden of chosing iteration parameters. It is quite difficult to determine the optimal parameter(s) in both p-cyclic SOR and block AOR. In the case of 3-AOR, it may be hard to select parameters for which the algorithm even converges. If anything, the experiments may understate the advantage of the conjugate gradient methods, since production codes would normally need to determine the SOR and AOR parameters adaptively.

By establishing that algorithm BNP is the limiting case of the preconditioned weighting method, we have shown that BNP is competitive with yet another approach to solving problem LSE. Again, the experimental evidence supports the analysis. For each of the problems we considered, algorithm INMG, which is essentially an improved version of BNP, achieved performance comparable to (and generally faster than) preconditioned weighting.

Perhaps the most encouraging aspect of this work is the characterization of algorithm BNP as a nullspace method, and the resulting extension to more general implicit nullspace methods. We obtain a class of methods capable of solving problems for which algorithm BNP does not appear to be well suited, including saddle point problems, and problems in which G lacks full column rank. Moreover, for problems reflecting a substructuring of the physical domain, implicit nullspace methods offer opportunities for parallel computation.

In addressing the question of how to construct an augmentation matrix which leads to an effective implicit nullspace method, we have merely scratched the surface: of the methods we have tested, only those which depend heavily on rows of G (algorithms INMG and INMGI) show potential as truly robust algorithms. But flexibility in the choice of the augmentation matrix M_1 appears to be the greatest strength of the approach we describe. We strongly believe that further research on more effective ways to generate the augmentation matrix will prove fruitful.

8. References

[1] J. Batt and S. Gellin, *Rapid Reanalysis by the Force Method*, Comp. Meth. in App. Mech. and Eng., 53 (1985), pp. 105–117.

[2] J. Barlow, *Error analysis and aspects of deferred correction for equality constrained least squares problems*, SIAM J. Numer. Anal., 25 (1988), pp. 1340–1358.

[3] ——, private communication, December, 1989.

[4] J. Barlow, N. Nichols and R. Plemmons, *Iterative methods for equality constrained least squares problems*, SIAM J. Sci. Statist. Comp., 9 (1988), pp. 892–906.

[5] ——, *Iterative methods for equality constrained least squares problems*, Internal Report CS-87-04, Dept. of Computer Science, Pennsylvania State University, January, 1987.

[6] M. Berry, M. Heath, I. Kaneko, M. Lawo, R. Plemmons and R. Ward, *An Algorithm to Compute a Sparse Basis of the Null Space*, Numer. Math., 47 (1985), pp. 483-504.

[7] A. Bjorck, Least Squares Methods, in Handbook for Numerical Methods, ed. by P. Ciarlet and J. Lions, Elsevier/North Holland Vol. 1, 1989.

[8] J. Dongarra, J. Bunch, C. Moler and G. Stewart, LINPACK Users' Guide, SIAM, Philadelphia, PA, 1979.

[9] I. Duff, R. Grimes, J. Lewis, *Users' Guide for the Harwell-Boeing Sparse Matrix Collection*, manuscript, Boeing Computer Services, Seattle, WA, 1988.

[10] M. Eiermann, W. Niethammer and A. Ruttan, *Optimal successive overrelaxation iterative methods for p–cyclic matrices*, Numer. Math., to appear.

[11] R. Freund, *A note on two block-SOR methods for sparse least squares problems*, Lin. Alg. and Applic., 88–89 (1987), pp. 211–221.

[12] G. Golub and C. van Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, MD, 1983.

[13] A. Hadjidimos, *Accelerated overrelaxation method*, Math. Comp., 32 (1978), pp. 149–157.

[14] L. Hageman and D. Young, Applied Iterative Methods, Academic Press, New York, NY, 1981.

[15] C. Hall, *Numerical solution of Navier-Stokes problems by the dual variable method*, SIAM J. Alg. Disc. Meth., 6 (1985), pp. 220–236.

[16] F. Harlow and F. Welch, *Numerical calculations of time dependent viscous incompressible flow of fluid with a free surface*, Phys. Fluids, 8 (1965).

[17] M. Heath, R. Plemmons and R. Ward, *Sparse Orthogonal Schemes for Structural Optimization Using the Force Method*, SIAM J. Sci. Statist. Comp., 5 (1984), pp. 514–532.

[18] D. James, *Implicit Nullspace Iterative Methods for Constrained Least Squares Problems*, submitted to SIAM J. Sci. Statist. Comp, Dec. 1989.

[19] ——, *Order-Reducing Conjugate Gradients versus Block AOR for Constrained Least Squares Problems*, Lin. Alg. and Applic., to appear, 1990.

[20] D. James and R. Plemmons, *An Iterative Substructuring Algorithm for Equilibrium Equations*, Numer. Math., to appear, 1990.

[21] A. Jennings and M. Ajiz, *Incomplete Methods for Solving $A^TAx = b$*, SIAM J. Sci. Statist. Comp., 5 (1984), pp. 978–987.

[22] D. Kershaw, *The incomplete Choleski-conjugate gradient method for iterative solution of systems of linear equations*, J. Comp. Phys., 26 (1978), pp. 42–65.

[23] T. Manteuffel, *Shifted Incomplete Cholesky Factorization*, in Sparse Matrix Proceedings 1978, I. Duff and G. Stewart, eds., SIAM, Philadelphia, PA, 1979.

[24] T. Markham, M. Neumann, and R. Plemmons, *Convergence of a direct-iterative method for large scale least squares problems*, Lin. Alg. and Applic., 69 (1985), pp. 155–167.

[25] J. Meijerink and H. van der Vorst, *An Iterative Solution Method for Linear Systems of Which the Coefficient Matrix is a Symmetric M-Matrix*, Math. Comp., 31 (1977), pp. 148–162.

[26] J. Ortega, Introduction to Parallel and Vector Solution of Linear Systems, Plenum Press, New York, NY, 1988.

[27] J. Ortega and R.Voigt, *Solution of Partial Differential Equations on Vector and Parallel Computers*, SIAM Review, 27 (1985), pp. 149–240.

[28] E. Papadopoulou, Y. Saridakis and T. Papatheodorou, *Block AOR Iterative Schemes for Large-Scale Least-Squares Problems*, SIAM J. Numer. Anal., 26 (1989), pp. 637-660.

[29] D. Pierce., A. Hadjidimos and R. Plemmons, *Optimality relationships for p–cyclic SOR*, Numer. Math., 56 (1990), pp. 635–643.

[30] R. Plemmons, *A Parallel Block Iterative Scheme Applied to Computations in Structural Analysis*, SIAM J. Alg. Disc. Meth., 7 (1986), pp. 337-347.

[31] R. Plemmons and R. White, *Substructuring methods for computing the nullspace of equilibrium matrices*, SIAM J. on Matrix Anal. and Applic., 11 (1990), pp. 1–22.

[32] J. Przemieniecki, Theory of Matrix Structural Analysis, Dover Publications, Inc, 1985.

[33] Y. Saad, *Preconditioning techniques for nonsymmetric indefinite linear systems*, J. Comp. App. Math., 24 (1988), pp. 89–105.

[34] G. Strang, *A framework for equilibrium equations*, SIAM Review, 30 (1988), pp. 283–297.

[35] ——, Introduction to Applied Mathematics, Wellesley Cambridge Press, Wellesley, MA, 1986.

[36] R. Varga, *p–cyclic matrices: a generalization of the Young-Frankel successive overrelaxation scheme*, Pacific J. Math., 9 (1959), pp. 617–623.

[37] ——, Matrix Iterative analysis, Prentice-Hall, Englewood Cliffs, NJ, 1963.

[38] D. Young, Iterative Solution of Large Linear Systems, Academic Press, New York, NY, 1971.

[39] ——, *Iterative methods for solving partial differential equations of elliptic type*, Trans. Amer. Math. Soc., 76 (1954), pp. 92–111.

[40] Z. Zlatev and H. Nielsen, *Solving large and sparse linear least-squares problems by conjugate gradient algorithms*, Comput. Math. Appl., 15 (1988), pp. 185–202.

9. Appendix: The Breakdown of Incomplete QR Factorizations

The research which led to this dissertation began with a look at incomplete QR preconditioners for ordinary least squares problems. We quickly developed an interest in order-reducing conjugate gradients for constrained problems, leaving unfinished the work on incomplete QR factorizations. Before this change in direction, however, we encountered some simple examples of a mechanism which can cause incomplete QR factorizations to break down. We felt these examples were interesting enough to deserve a separate discussion; we include them in this appendix. The notation throughout the appendix is independent of the preceding chapters.

9.1 Preconditioning with Incomplete Factorizations

We consider the following ordinary least squares problem: given an $m\times n$ matrix A with full column rank, and an $m\times 1$ vector b,

$$\text{minimize} \quad \|Ax - b\|_2. \tag{9.1}$$

The unique solution x satisfies the so-called normal equations:

$$A^T Ax = A^T b. \tag{9.2}$$

Notice that the coefficient matrix $A^T A$ is symmetric positive definite when A has full column rank.

There are, however, a number of well-known difficulties associated with using the normal equations (see, for example, Bjorck [7] or Golub and van Loan [12]). These include the conditioning of the problem (the spectral condition number of $A^T A$ is the square of the condition number of A itself), and the $O(n^3)$ cost and loss of information associated with forming $A^T A$. We can avoid some of these problems by leaving the normal equations in factored form and solving the problem iteratively (using, for example, a conjugate gradient algorithm). But convergence of the conjugate gradient algorithm still depends on the condition number of $A^T A$, so we generally need to precondition the normal equations to have any hope of producing an effective algorithm.

Now suppose we have an orthogonal factorization of A given by $A = QR$, where the matrix Q is orthogonal, and R is upper triangular. Recall that $R^T R = A^T A$; in fact, R is (up to changes in the signs of the rows) precisely the Cholesky factor of $A^T A$. Given such a factorization, we could "precondition" the normal equations:

$$R^{-T} A^T A R^{-1} w = R^{-T} A^T b, \quad \text{where } w = Rx. \tag{9.3}$$

But $AR^{-1} = Q$, and $Q^T Q = I$, so this reduces to $w = R^{-T} A^T b$.

Of course, there is no need to iterate on such a trivial system; we have in fact described a way to use a QR factorization to solve a least squares problem directly. But if we did apply a conjugate gradient algorithm, we'd achieve "convergence" in one iteration. Thus, in a sense, the Cholesky factor of $A^T A$ is the "perfect" preconditioner, and the thought experiment motivates an approach to preconditioning the normal equations (9.2). Let $\hat{R}$ be some non-singular upper triangular matrix which is in some sense a rough approximation of the Cholesky factor R of A^TA. We are tempted to ask whether $\hat{R}$ may be a good preconditioner for the normal equations: perhaps $A\hat{R}^{-1}$ is "roughly" orthogonal, and the matrix $\hat{R}^{-T}A^TA\hat{R}^{-1}$ is a crude approximation of the identity matrix I. If so, then the preconditioned system

$$\hat{R}^{-T}A^TA\hat{R}^{-1}\hat{w} = \hat{R}^{-T}A^Tb \quad \text{where } \hat{w} = \hat{R}x \tag{9.4}$$

ought to be well-suited for the conjugate gradient algorithm.

9.2 A Strategy for Incomplete QR

One way to form the matrix $\hat{R}$ is to use a so-called **incomplete Cholesky factorization** (see, for example, Meijerink and van der Vorst [25], or Ortega [26]). Such factorizations compute a conventional Cholesky factorization of a symmetric positive definite matrix, but retain only selected non-zeros as the factorization proceeds. One could use a strategy based on the *size* of the non-zeros, keeping only elements larger than some fixed or varying drop tolerance. Alternatively, one might instead employ a method based on the *position* of the non-zeros; here, one selects a predetermined non-zero pattern for the approximate Cholesky factor $\hat{R}$, preserving only those non-zeros which conform to the selected pattern. More sophisticated approaches, including combinations of these methods, are of course possible (see, for example, Manteuffel [23] for an algorithm which includes a shifting of the coefficient matrix).

The incomplete Cholesky factorization is certainly a promising approach to preconditioning symmetric positive definite systems *when the coefficient matrix* *is already explicitly available.* For problems involving normal equations, though, we generally want to avoid forming A^TA. For that reason, we might take a somewhat different approach to constructing the preconditioner $\hat{R}$: we could apply orthogonal rotations to A as if we were accomplishing a QR factorization, but again preserve only selected non-zeros. As with incomplete Cholesky, we can accomplish such an **incomplete QR factorization (IQR)** by using drop tolerances, specified non-zero patterns, or other more elaborate strategies (see, for example, Jennings and Ajiz [21], or Zlatev and Nielsen [40]).

Regardless of the drop strategy, there is a question of existence to address. Certainly a complete orthogonal factorization in exact arithmetic will produce a non-singular upper triangular matrix R. But once we begin dropping non-zeros, we may in fact introduce linear dependence in the columns of the reduced matrix, preventing us from generating a non-singular preconditioner $\hat{R}$. Jennings and Ajiz [21] describe a sophisticated strategy based on drop tolerances, and prove that it produces a non-singular preconditioner. They produce a similar result for a strategy based on Gram-Schmidt orthogonalization (see also Saad [33]). We examine here a strategy which uses Givens rotations and a drop strategy based on the position of non-zeros.

Begin by agreeing to confine non-zeros in the upper triangular matrix $\hat{R}$ to those positions in which A^TA has non-zeros. We determine the non-zero pattern of A^TA by symbolic matrix multiplication. More precisely, columns i and j of the matrix A are said to be **symbolically interactive** (or simply interactive) if there exists a k such that both a_{ki} and a_{kj} are non-zero. In this case, the product A^TA is said to have a symbolic non-zero in position (i, j). If no such k exists, we call the columns **symbolically non-interactive** (or simply non-interactive), and we note that A^TA has a (symbolic) zero in position (i,j). This approach to defining the non-zero pattern of A^TA (and $\hat{R}$) amounts to assuming that no cancellation occurs in forming the product. Of course, since $\hat{R}$ is upper triangular (and A^TA is symmetric positive definite), we need only consider the case $i < j$ when performing the symbolic multiplication. The diagonal elements of the product are always symbolically non-zero (in fact they are numerically positive), because the columns of A interact with themselves.

Now apply the following reduction to A:

Incomplete QR Factorization

1. Use symbolic matrix multiplication to determine the non-zero data structure of the upper triangular portion of A^TA. Use this information to fix a static data structure for the approximate Cholesky factor $\hat{R}$.

2. For **targetrow** $= 1, \ldots, n$:

   (a) For **pivotrow** $= 1, \ldots,$ (**targetrow**-1):

   - If necessary, rotate **pivotrow** against **targetrow**, annihilating the leading non-zero in position $k =$ **pivotrow** of **targetrow**.
   - Accumulate all fill in **targetrow**; *i.e.* drop no non-zeros produced in **targetrow** by the Givens rotation.
   - Retain non-zeros in **pivotrow** only if they occur in positions corresponding to symbolic non-zeros in the upper triangular portion of A^TA.

   (b) Now that **targetrow** has been completely reduced, retain only those non-zeros which occur in positions corresponding to symbolic non-zeros in the upper triangular portion of A^TA. Store the selected non-zeros of this fully reduced row in the static data structure reserved for $\hat{R}$.

Thus we allow complete fill in a target row while reducing that row, but restrict fill in the pivot rows to positions corresponding to symbolic non-zeros of A^TA.

In addressing whether this algorithm successfully produces a non-singular upper triangular $\hat{R}$, one might first ask the same question about incomplete Cholesky factorizations. It has long been known that incomplete Cholesky may break down when the drop strategy is based on the non-zero pattern of A^TA; Kershaw [22] provides a simple example. If A^TA is either diagonally dominant or an M-matrix, however, this form of incomplete Cholesky always produces a non-singular preconditioner (in fact, a slightly more general result holds; see Manteuffel [23]). One might be led to expect similar results for IQR. We demonstrate in the next section that this is not the case: the IQR algorithm described above can in fact break down, even if A^TA is diagonally dominant or an M-matrix.

9.3 Examples of Incomplete QR Breakdown

In this section we demonstrate by example that the IQR algorithm described above can in fact break down, even under fairly restrictive conditions.

Begin with the following 4×4 matrix A:

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 6 & 0 & 2 \end{bmatrix}. \tag{9.5}$$

The matrix has full column rank, so A^TA is symmetric positive definite. (A is square merely to keep the example small; one could easily construct rectangular matrices to achieve the effects described below.)

We now apply the IQR factorization described in the previous section. Forming the symbolic product A^TA gives us the non-zero pattern for $\hat{R}$:

$$\hat{R} = \begin{bmatrix} * & * & * & * \\ & * & 0 & * \\ & & * & * \\ & & & * \end{bmatrix}. \tag{9.6}$$

Note that there are symbolic non-zeros in every position except $(2,3)$.

We now detail the reduction. The non-zeros in row 1 of A conform to the required non-zero pattern, so we retain row 1 as is. Row 2 has a zero in column 1, so no reduction is necessary. Again, its non-zeros conform to the required pattern, so we retain all the non-zeros. Now let row 3 be the target row. Begin by using row 1 as the pivot row to annihilate the $(3,1)$ element. Applying the appropriate Givens rotation produces

$$\begin{array}{r} \text{pivot} \rightarrow \\ \\ \text{target} \rightarrow \\ \\ \end{array} \begin{bmatrix} 1.4142 & 0.7071 & 0.7071 & 0.7071 \\ 0 & 1.0000 & 0 & 1.0000 \\ 0 & -0.7071 & 0.7071 & 0.7071 \\ 0 & 6.0000 & 0 & 2.0000 \end{bmatrix}. \tag{9.7}$$

Our strategy calls for preserving all elements in the target row until the reduction of that row is complete, so at this point we discard no elements in row 3. The non-zeros in the pivot row conform to the required pattern, so we preserve all elements there as well. Thus, up until now, our "incomplete" factorization coincides with a classical QR factorization. We are not finished reducing row 3, however. Now use row 2 as the pivot row to annihilate the $(3,2)$

element. The complete Givens rotation (no discarded elements) looks like this:

$$\begin{array}{r} \\ \text{pivot} \rightarrow \\ \text{target} \rightarrow \\ \\ \end{array} \left[\begin{array}{cccc} 1.4142 & 0.7071 & 0.7071 & 0.7071 \\ 0 & 1.2247 & -0.4082 & 0.4082 \\ 0 & 0 & 0.5774 & 1.1547 \\ 0 & 6.0000 & 0 & 2.0000 \end{array} \right]. \tag{9.8}$$

Recall, however, that there is a symbolic zero in position (2,3), so we must discard the (2,3) element in the pivot row (in practice, we do not even generate it). We also note that the reduction of target row 3 is now complete, so we check to see if we must discard any non-zeros. The non-zeros in row 3 conform to the required non-zero pattern, so we preserve all the non-zeros in this row. Thus, when the reduction of target row 3 is complete, the matrix $\hat{R}$ looks like this:

$$\begin{array}{r} \\ \\ \text{target} \rightarrow \\ \\ \end{array} \left[\begin{array}{cccc} 1.4142 & 0.7071 & 0.7071 & 0.7071 \\ 0 & 1.2247 & 0 & 0.4082 \\ 0 & 0 & 0.5774 & 1.1547 \\ 0 & 6.0000 & 0 & 2.0000 \end{array} \right]. \tag{9.9}$$

Discarding the (2,3) entry has created a dependency which will become apparent on the next rotation. Row 4 now becomes the target row; using pivot row 2 to annihilate the leading non-zero produces the singular matrix below:

$$\begin{array}{r} \\ \text{pivot} \rightarrow \\ \\ \text{target} \rightarrow \end{array} \left[\begin{array}{cccc} 1.4142 & 0.7071 & 0.7071 & 0.7071 \\ 0 & 6.1237 & 0 & 2.0412 \\ 0 & 0 & 0.5774 & 1.1547 \\ 0 & 0 & 0 & 0 \end{array} \right]. \tag{9.10}$$

The algorithm has failed to produce a non-singular upper triangular $\hat{R}$.

It's helpful to examine how we constructed the example. The 3×3 principal submatrix in the upper left corner of (9.10) is non-singular, so its columns span $\mathcal{R}^3$. In particular, the vector consisting of the first three entries of the last column of (9.10) can be written as a unique linear combination of the columns of this submatrix. Thus

$$\alpha_1 \begin{bmatrix} 1.4142 \\ 0 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0.7071 \\ 6.1237 \\ 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} 0.7071 \\ 0 \\ 0.5774 \end{bmatrix} = \begin{bmatrix} 0.7071 \\ 2.0412 \\ 1.1547 \end{bmatrix} \tag{9.11}$$

for some unique choice of α_1, α_2, and α_3. Now choose the elements in the fourth row of the original matrix A so that the entries in the row satisfy the same relationship:

$$\alpha_1 a_{41} + \alpha_2 a_{42} + \alpha_3 a_{43} = a_{44}. \tag{9.12}$$

This ensures the matrix lacks full column rank *after* the element in position $(2, 3)$ is discarded. If we choose the entries in this row so that (9.12) is satisfied and the *original* matrix has full column rank, we succeed in producing an example of breakdown.

Note that this construction is not overly dependent on the fact that we are using the non-zero pattern of $A^T A$; given another strategy based on a specified non-zero pattern, we may be able to construct an example of breakdown in a similar manner. We need only begin to reduce a simple example, note the stage at which the first element is discarded, and adjust the values of the elements below the current target row to create a column dependency. (We should also observe in passing that a dropped element can have the opposite effect, changing a dependent set of vectors into an independent set.)

We note several other interesting facts about the algorithm and the breakdown mechanism:

1. *Row permutations may or may not prevent breakdown.* If we permute rows 2 and 4 in the matrix A above, the incomplete factorization terminates normally. But there is nothing special about this permutation: rows 2

and 4 have the same non-zero pattern, and we could just as easily have chosen numerical values to cause breakdown for this permutation.

2. *There is at least one situation which guarantees a successful factorization.* Suppose we are given a matrix A with many more rows than columns, and we partially reduce the matrix until reaching the form

$$\begin{bmatrix} R_1 \\ A_2 \end{bmatrix}, \tag{9.13}$$

where R_1 is upper triangular and non-singular. Then subsequent incomplete factorization cannot create dependent columns (in fact, continued reduction cannot reduce the singular values of R_1). This is because a Givens rotation which uses a leading non-zero r in a pivot row of R_1 to annihilate a leading non-zero a in a target row of A_2 will always leave the larger non-zero value $\sqrt{r^2 + a^2}$ on the diagonal of the pivot row. This fact could prove useful in problems involving *updating*: if we start with a matrix A and an approximate $\hat{R}$, then introduce new observations (rows), we can update $\hat{R}$ using IQR without fear of breakdown.

3. *We can place much stronger conditions on the matrix A and still experience breakdown.* Consider, for example, the matrix

$$A = \begin{bmatrix} 9 & -3 & 0 & 0 \\ 0 & 9 & 0 & -12 \\ 3 & 0 & 9 & 3 \\ 0 & -7 & 0 & 9 \end{bmatrix}. \tag{9.14}$$

In this case, $A^T A$ is strictly diagonally dominant, but breakdown occurs exactly as it did in the example above. Similarly, the matrix

$$A = \begin{bmatrix} -5 & 5 & 0 & 10 \\ 0 & 5 & 0 & -20 \\ 5 & 0 & -5 & 5 \\ 0 & -3 & 0 & 5 \end{bmatrix} \tag{9.15}$$

leads to a breakdown, despite the fact that A^TA is an M-matrix.

Finally, we report that breakdowns such as these do in fact occur in practical problems. We have tested the IQR factorization described in this section on the four sparse rectangular matrices WELL1033, WELL1850, ILLC1033, and ILLC1850 from the Harwell-Boeing test collection (see Duff, Grimes, and Lewis [9]), and observed breakdowns on WELL1850 and ILLC1850. We were able to prevent breakdown by reordering the rows of the matrices in stairstep fashion according to the positions of the leading non-zeros. Unfortunately, though, simply producing a non-singular $\hat{R}$ is not enough. For each of the four matrices we tested, the quality of the IQR preconditioner was marginal at best: we have not yet produced results which are fast enough to be competitive with any of the established iterative methods.